UNIVERSIDAD DE VALLADOLID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN.

GUIADO GNSS DE TRACTORES; GENERACIÓN DE CURVAS PARALELAS.

SEPTIEMBRE 2024

AUTOR: GABINO MARTÍNEZ GARCÍA

TUTOR: JAIME GÓMEZ GIL

TITULO: Guiado GNSS de tractores; generación de curvas paralelas.

AUTOR: Gabino Martínez García

TUTOR: Jaime Gómez Gil, Departamento de Teoría de la Señal, Comunicaciones e Ingeniería Telemática, E.T.S.I. Telecomunicación, Universidad de Valladolid.

DEPARTAMENTO: Departamento de Teoría de la Señal, Comunicaciones e Ingeniería Telemática.

Miembros del Tribunal

PRESIDENTE: Jaime Gómez Gil.

SECRETARIO: Juan Blas Prieto.

VOCAL: Alonso Alonso Alonso

SUPLENTE: Javier Aguiar Pérez

FECHA DE LECTURA: 27 de septiembre de 2024

CALIFICACIÓN:

Resumen

Las tractores realizan las tareas agrícolas en parcelas mediante sucesivas pasadas. Normalmente, los agricultores comienzan en una linde de la parcela que sea recta, y tratan toda la parcela mediante líneas rectas. En ocasiones, ninguna de las lindes de la parcela son rectas, y los agricultores tienen que tratar la parcela mediante trayectorias curvas. Los tractores realizan un mejor trabajo cuando las trayectorias son curvas. Este trabajo tiene como objetivo generar curvas paralelas para definir la trayectoria de los tractores en parcelas agrícolas que acaben tendiendo a una recta tras varias pasadas. Para ello se han estudiado distintos métodos de generación de curvas paralelas, tanto matemáticos, que ofrecen una mayor libertad de diseño, pero aparece una cierta complejidad geométrica, como mediante librerías de programación como, *Shapely* para Python y *GEOS* para C, que ofrecen opciones de paralelización más directas, pero menos configurables. Se ha presentado un algoritmo de generación de paralelas a partir de vector normal que ha sido combinado con un método de enderezado. El algoritmo combinado ha sido implementado en una aplicación desarrollada en MATLAB, la cual permite ajustar varios parámetros para generar diferentes configuraciones para la generación de las paralelas. Se generaron paralelas de cuatro curvas reales tomadas en una parcela en Pozal de Gallinas con diferentes parámetros. En las diferentes configuraciones se variaron parámetros como, el solape máximo, el espacio sin tratar o el número de paralelas. Surgió el problema de la aparición de cúspides, pero se solucionó mediante variaciones de valores de los parámetros. Los resultados conseguidos a partir de curvas reales muestran que, en estas parcelas y con condiciones de solape y de zona sin tratar habituales, la aplicación es capaz de generar paralelas tendiendo a una recta, llegando a una trayectoria casi recta en 20 pasadas.

PALABRAS CLAVE: Curvas paralelas, Cúspides, Enderezado, Solape, Sin tratar, MATLAB, appDesigner, vector, matriz.

Agradecimientos

Agradecer a mis abuelos, a mis padres y a mi hermana por el apoyo que siempre me han brindado no solo durante los cuatro años de la carrera sino durante toda mi vida.

También agradecer al Dr. D. Jaime Gómez Gil por guiarme y enseñarme estos meses de trabajo.

Además, me gustaría agradecer al Dr. Sergio Alonso García y a su hermano Ángel Alonso García por proporcionarme las curvas que he usado para mostrar los resultados, así como por su disponibilidad en las diversas reuniones que hemos realizado.

Por último, agradecer al Dr. Cesar Palencia de Lara por su ayuda en la creación del algoritmo de enderezado.

Muchas gracias a todos.

Índice abreviado

Índice general

Índice de figuras

ÍNDICE DE FIGURAS

ÍNDICE DE FIGURAS

Capítulo 1: Teoría sobre generación de curvas paralelas

Capítulo 1: Teoría sobre generación de curvas paralelas

La conducción de vehículos agrícolas dentro de parcelas que sean rectangulares o cuadradas permite realizar las diferentes pasadas en línea recta. Cuando los límites de la parcela adquieren una forma más irregular suele ser necesario realizar también pasadas curvas. En estos casos puede que no se optimice de la mejor forma el recorrido. Para aprovechar mejor el espacio existe la generación de curvas paralelas de forma que se consiguen distancias constantes. La generación de estas curvas no es un asunto trivial ya que este proceso puede provocar discontinuidades en las curvas producidas.

1.1 Ámbito del proyecto

Las tractores realizan las tareas agrícolas en parcelas mediante sucesivas pasadas. Normalmente, los agricultores comienzan en una linde de la parcela que sea recta, y tratan toda la parcela mediante líneas rectas. En ocasiones, ninguna de las lindes de la parcela son rectas, y los agricultores tienen que tratar la parcela mediante trayectorias curvas. Los tractores realizan un mejor trabajo cuando las trayectorias son rectas (ver **Figura 1.1)**

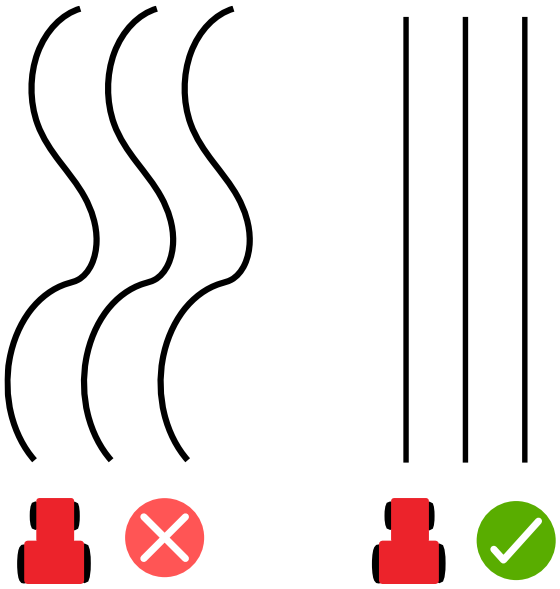

Figura 1.1. Los tractores realizan un mejor trabajo cuando las trayectorias son rectas. Es preferible realizar el recorrido de la derecha antes que el de la izquierda.

1.2 Objetivos

El objetivo principal de este Trabajo Final de Grado es la búsqueda, diseño e implementación de un algoritmo de generación de curvas paralelas con la finalidad de aplicarlo al guiado de tractores. Se

busca que tras varias pasadas acabe tendiendo a una recta. Este objetivo general se ha descompuesto en los siguientes objetivos concretos.

1- Estudiar teoría sobre generación de curvas paralelas. Analizar en profundidad los métodos matemáticos y algorítmicos que permiten generar curvas paralelas.
2- Implementar una aplicación que genere curvas paralelas y evaluar su funcionamiento. Utilizar los métodos estudiados para generar curvas paralelas
3- Mejorar la aplicación anterior en caso de que una implementación inicial no genere curvas de forma adecuada.
4- Conseguir que la aplicación pueda enderezar las curvas, dejando solo espacio sin tratar, dejando solo espacio de solape o permitiendo ambos espacios.
5- Evaluar la aplicación en curvas reales obtenidas en una parcela agrícola, y ver si tienden a trayectorias rectas.
6- Proponer mejoras futuras en la aplicación.

1.3 Fases y métodos

El proceso de creación y confección de este trabajo de fin de grado ha seguido las siguientes fases.

1. Estudio de la teoría asociadas a la generación de curvas paralelas. Se han estudiado métodos de generación puramente matemáticos y también métodos algorítmicos mediante librerías de programación.
2. Diseño de un algoritmo para la generación de paralelas y el enderezado de curvas.
3. Desarrollo de una aplicación que genere curvas paralelas y las enderece. Se ha creado en MATLAB. Posee varios parámetros configurables a través de una interfaz gráfica.
4. Evaluación del funcionamiento de la aplicación mediante el uso de curvas reales tomadas en una finca agrícola en Pozal de Gallinas. Se han paralelizado cuatro curvas variando los valores de los parámetros.

1.4 Organización de la memoria

La memoria se ha estructurado en cinco capítulos.

El capítulo 1 tiene como objetivo servir de introducción al trabajo.

El capítulo 2 se encarga de presentar métodos teóricos para la generación de curvas paralelas. Se exponen tanto métodos puramente matemáticos como procedimientos algorítmicos mediante el uso de lenguajes de programación, en concreto Python y C.

El capítulo 3 presenta de forma general el algoritmo que se utilizará para el enderezado de curvas y como este se combina con el método de paralelización.

El capítulo 4 es el más extenso y describe detalladamente tanto la interfaz de la aplicación como el código que se ha creado para la implementación del algoritmo. En este capítulo también se presentan los resultados de la aplicación, es decir, diferentes curvas paralelizadas utilizando varios valores para los parámetros.

El capítulo 5 recoge las principales conclusiones del trabajo, así como las posibles líneas de investigación futuras.

Capítulo 2:
Teoría sobre la generación de curvas paralelas

Capítulo 2: Teoría sobre generación de curvas paralelas

En este capítulo se exponen los conceptos teóricos para la generación de curvas paralelas. Se van a presentar métodos de generación matemáticos [2]. También en este capítulo se van a exponer algoritmos de paralelización en los lenguajes de programación Python y C mediante el uso de las librerías *Shapely* y *GEOS* respectivamente.

2.1 Curva paralela por definición

Una curva paralela principalmente se puede definir de dos formas. La primera define una curva paralela como aquella cuyos puntos se encuentran todos a la misma distancia de los de la curva original en la dirección normal. La segunda define una curva paralela como la envolvente de una serie de circunferencias de mismo radio centradas en la curva original.

2.2 A partir del vector normal

Utilizando la primera definición, Una curva paralela se define como $\overrightarrow{x_d}(t) = \vec{x}(t) + d \cdot \vec{n}(\mathrm{t})$, siendo *d* la distancia respecto a la curva original, $\vec{n}$ el vector normal unitario en forma paramétrica y una curva en forma paramétrica tal que $\vec{x}(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix}$,siendo x(t) e y(t) las dos coordenadas paramétricas

El vector $\vec{n}$ tiene la forma $\frac{1}{|\vec{x}'|} \cdot \begin{bmatrix} -y' \\ x' \end{bmatrix}$, siendo $|\vec{x}'| = \sqrt{x'^2 + y'^2}$ el módulo de la derivada $\vec{x}(t)$ para hacer el vector resultante unitario. Partiendo desde $\begin{bmatrix} x(t) \\ y(t) \end{bmatrix}$, se calcula el vector tangente haciendo la derivada respecto a *t* de las dos componentes $x' = \frac{dx(t)}{dt}$, $y' = \frac{dy(t)}{dt}$ y posteriormente se rota el vector resultante 90 grados en alguna dirección tal que el resultado es $\begin{bmatrix} -y' \\ x' \end{bmatrix}$.

Además de la formula general también existe otra fórmula que especifica cada coordenada paramétrica de la curva paralela por separado tal que:

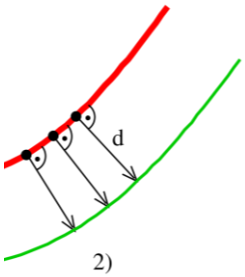

Figura 2.1 Una curva y su paralela donde se observa el método de paralelización a partir del vector normal. La curva original es la roja y la paralela es la verde.[3]

$$x_d(t) = x(t) + \frac{d \cdot y'}{|\vec{x}'|}$$
$$y_d(t) = y(t) + \frac{d \cdot x'}{|\vec{x}'|} \qquad (1)$$

Pudiendo ser *d* tanto negativa como positiva de forma que la curva paralela aparecerá por debajo o por encima de la curva original respectivamente.

2.3 A partir de una familia de círculos congruentes

Utilizando la segunda definición, es necesario conocer cómo se calcula la envolvente de una serie de circunferencias congruentes. Para ello, se indicará como calcular la envolvente de curvas planares de forma general y posteriormente se particularizará para nuestro caso de interés.

Para una familia de curvas planares, la envolvente es aquella curva que es tangencial a todas las curvas que componen la familia en un plano. La misma definición se podría extrapolar para la envolvente de una superficie.

Considerando una familia de curvas planares caracterizadas por un parámetro β, tenemos que las curvas serán función de *x, y, β*. Siendo entonces una familia de curvas genéricas $f(x, y, \beta)$, las ecuaciones paramétricas de la envolvente vienen definidas por el sistema de ecuaciones (2)

$$\begin{cases} f(x, y, \beta) = 0 \\ \dfrac{\partial}{\partial \beta} \cdot f(x, y, \beta) = 0 \end{cases} \qquad (2)$$

Eliminando el parámetro β se puede obtener la ecuación implícita y explicita de la envolvente. Es importante destacar que el sistema de ecuaciones indicado es condición necesaria para la existencia de la envolvente, pero no suficiente. Este sistema de ecuaciones puede dar como solución otros puntos singulares que no pertenecen a la envolvente. Para corroborar que todos los puntos de la función resultante pertenecen a la envolvente existen dos condiciones suficientes que se deben cumplir: (3)

$$\begin{vmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \\ \frac{\partial}{\partial x}\left(\frac{\partial}{\partial \beta} f\right) & \frac{\partial}{\partial y}\left(\frac{\partial}{\partial \beta} f\right) \end{vmatrix} \neq 0 \,, \quad \frac{\partial^2 f}{\partial \beta^2} \neq 0 \qquad (3)$$

Una vez conocido el método para calcular envolventes de curvas generales nos vamos al caso que nos atañe, familia de círculos congruentes. La ecuación de un círculo de radio 1 y centro (1, 1), tiene la forma $(x+1)^2 + (y+1)^2 = 1$. Conociendo esto, si en lugar de (1, 1), se define el centro como una ecuación paramétrica de la forma $\begin{bmatrix} \beta \\ f(\beta) \end{bmatrix}$ tal que modela la curva original sobre la que se quieren

hacer los paralelos, se tendrán una serie de círculos de radio 1 cuyos centros se desplazarán según la función $f(\beta)$. La función tendrá la forma $(x+\beta)^2+(y+f(\beta))^2=1$ que corresponde a una familia de círculos de mismo radio. Si calculamos la envolvente como se indicó anteriormente, se obtendrá la ecuación de la curva paralela a la original por la propia definición de curva paralela. Si se dan distintos valores de radio en lugar de 1, siendo este radio *d* constante respecto a β, se obtendrán curvas paralelas a la original a diferentes distancias.

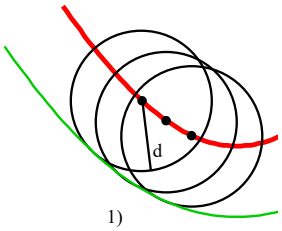

Figura 2.2. Una curva y su paralela donde se observa el método de paralelización a mediante círculos congruentes. La curva original es la roja y la paralela es la verde.[3]

2.4 Propiedades Geométricas

Es importante destacar que las curvas paralelas no tienen las mismas propiedades geométricas que las curvas originales, de hecho, excepto en el caso del círculo o la particularización en las líneas paralelas las expresiones matemáticas que la caracterizan suelen ser más complicadas. Dependiendo del tipo de expresión y de la distancia entre curvas pueden aparecer picos o incluso solapamiento y, partiendo de funciones originales racionales, la de la curva paralela no tiene por qué serlo.

2.5 Curvas implícitas

Normalmente el cálculo de una curva paralela en base a una curva implícita no es sencillo. Sin embargo, en algunos casos se puede describir utilizando una función distancia. Por ejemplo, en el caso del círculo: $f(x,y)=x^2+y^2-1=0$, tiene como función de distancia $h(x,y)=\sqrt{x^2+y^2}-1=d$.

Una curva paralela a una distancia d es el conjunto de nivel $h(x,y)=d$ de la función de distancia orientada correspondiente.

2.5.1 Propiedades Geométricas

Presuponiendo una serie de condiciones se puede probar la existencia de una función de distancia orientada $h(x,y)$. Las condiciones son las siguientes:

El módulo del gradiente de la función $h(x,y)$ debe de ser igual a 1.

La función $h((x,y)+d*\nabla h(x,y))$, la función *h* en función de x, y más la distancia entre curvas por el gradiente de la función *h*, debe ser igual que la función $h(x,y)$más la distancia entre curvas $h(x,y)+d$.

En base a la condición anterior tenemos que, el gradiente de la función $h((x,y)+d*\nabla h(x,y))$ debe ser igual al gradiente de $h(x,y)$.

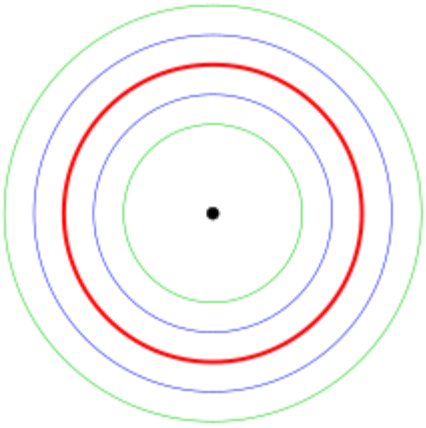

Figura 2.3 El círculo y sus paralelas definidas por la función de distancia $h(x,y) = \sqrt{x^2+y^2} - 1 = d$. *El círculo original es el rojo. Esta figura ilustra cómo se pueden obtener curvas paralelas con distancias constantes respecto al círculo original, a partir de una fórmula, ejemplificando el proceso de paralelización en curvas implícitas. [4]*

2.6 Curvas paralelas de la librería Shapely para python

En el lenguaje de programación Python existe un paquete para geometría computacional llamado Shapely. Utiliza funciones de la librería de C/C++ GEOS. Su modelo espacial se compone principalmente de tres objetos geométricos fundamentales; puntos, curvas y superficies.

En Shapely se usa programación orientada a objetos de forma que los objetos geométricos son instancias creadas a partir de una clase con métodos y atributos, propios y comunes. De esta forma, por ejemplo, el punto se implementa con la clase *point*, una curva se puede implementar mediante las clases *LineString* o *LinearRing* y una superficie mediante la clase *polygon*. Se pueden crear colecciones de puntos, curvas y superficies mediante las clases *MultiPoint, MultiLineString* y *MultiPolygon* respectivamente.

El modelo espacial de Shapely considera tres "conjuntos" de puntos en el plano: interior, límite y exterior. Con estos conjuntos se caracterizan los puntos, curvas y superficies.

- **Punto**: Tiene un interior de un punto, no tiene límite y un exterior de todos los demás puntos del plano. Su dimensión topológica es 0.
- **Curva**: Tiene un interior formado por infinitos puntos a lo largo de su longitud, un límite formado por los puntos que se encuentran en los dos extremos de la curva y un exterior de todos los demás puntos del plano. Su dimensión topológica es 1.
- **Superficie**: Tiene un interior formado por infinitos puntos, un límite formado por una o más curvas y un exterior de todos los demás puntos del plano (incluidos los posibles "agujeros" que existan). Su dimensión topológica es 2.

Shapely hace una distinción entre métodos constructivos y métodos de teoría de conjuntos. Los métodos que usaremos para generar las curvas paralelas serán constructivos. En concreto nos centraremos en los métodos de "buffer" y "offset_curve. [1].

2.6.1 Método Buffer

El método calcula el buffer de un objeto geométrico para una distancia positiva o negativa. Se define como la suma o la diferencia **de Minkowski**, dependiendo si la distancia es positiva o negativa, de un objeto geométrico con un círculo cuyo radio es igual al valor absoluto del parámetro distancia. [5].

2.6.2 Suma de Minkowski

La suma de Minkowski [6] se define como el conjunto de los vectores generados de sumar todos los puntos a y b pertenecientes a las curvas planares O_1 y O_2 respectivamente. Definiendo el perímetro de las curvas planares O_1 y O_2 como C_1 y C_2, el problema de calcular la suma de Minkowski se convierte en el problema de calcular la convolución de estos dos perímetros. De esta forma cuando O_1 y O_2 son objetos convexos la convolución de C_1 y C_2 coincide con el perímetro de la suma de Minkowski.

La curva formada de la convolución de los perímetros puede tener algunas partes redundantes que no contribuyen al perímetro de la suma de Minkowski, por lo que, para construir este perímetro es necesario no solo calcular la convolución de C_1 y C_2 sino también eliminar las partes redundantes que no contribuyen.

Entonces llegamos a la conclusión donde radica el interés de estudiar esta suma para la generación de curvas paralelas. La convolución de C_1 y C_2 se puede considerar como la curva envolvente obtenida a partir de realizar un barrido de la curva C_1 a lo largo de la curva C_2. Si realizamos la convolución entre una curva planar cualquiera y un círculo con centro (0,0), obtenemos una curva que por la consideración indicada anteriormente respecto al cálculo de la convolución es, por definición, una curva paralela.

La curva paralela resultante no será una curva racional (en forma de polinomio) aunque se origine de dos curvas racionales. En nuestro caso, partimos de curvas muestreadas mediante puntos de las cuales no conocemos su expresión polinómica por lo que a priori no nos afecta la propiedad mencionada.

2.7 Curvas paralelas mediante la librería GEOS

GEOS (Geometry Engine Open Source) es una librería de C/C++ para geometría computacional centrada en algoritmos utilizados en software de Sistemas de Información Geográfica (SIG).

En concreto, en esta librería [7], existe un archivo de cabecera para C++ llamado <OffsetCurve.h> que posee funciones para generar curvas paralelas. Computa curvas paralelas a partir de una geometría. Si la distancia de desplazamiento de la curva es positiva, la curva se genera a la izquierda de la entrada. Si la distancia de desplazamiento es positiva se genera a la derecha.

La obtención de la curva paralela utiliza el límite del búfer de la geometría a la distancia de desplazamiento. El búfer positivo (o negativo) de una geometría se define como la suma (o diferencia) de Minkowski de la geometría con un círculo cuyo radio es igual al valor absoluto de la distancia del búfer.

A partir de la curva de desplazamiento sin procesar, se escogen las secciones del límite del buffer que se encuentran en esa curva, es decir, la intersección entre las dos curvas. Las secciones calculadas

están ordenadas a lo largo de la curva de desplazamiento sin procesar y son disjuntas, no tienen intersección consigo mismas, aunque pueden formar anillos.

En el modo joined las secciones computadas para cada línea de entrada se juntan en una sola curva paralela, la curva generada puede tener intersecciones consigo misma.

La obtención de curvas paralelas mediante este algoritmo presenta dos problemas, la producción de curvas desconectadas y la inclusión de segmentos innecesarios. Estos dos problemas se deben a la divergencia que hay entre la curva de desplazamiento sin procesar y el búfer. La divergencia hace que existan errores en el resultado por las pequeñas diferencias entre las curvas generadas Esta divergencia se produce por los algoritmos de generación del buffer que utilizan aproximación.

2.8 Conclusiones

En este capítulo se han presentado diferentes métodos teóricos para la generación de curvas paralelas. Los métodos descritos han sido tanto matemáticos como algorítmicos mediante el uso de librerías especializadas de lenguajes de programación. Se han utilizado *Shapely* y *GEOS*. Los diversos métodos tienen sus pros y sus contras. Los procedimientos matemáticos dan más libertad de diseño ya que permiten utilizar diferentes formas de realizar los cálculos necesarios, pero a su vez, son complejos de implementar por la necesidad de tener un cierto conocimiento de geometría. Los algoritmos mediante librerías tienen la ventaja de que ya están implementados, pero por ese mismo motivo, existe menos libertad debido a que su mejor o peor funcionamiento está mayormente definido, aunque permiten ciertos niveles de flexibilidad y personalización. Por ello para desarrollar la aplicación posteriormente se utilizará el algoritmo de generación de paralelas mediante vector normal que da más libertad de implementación.

Capítulo 3:
Cálculo de curvas paralelas

Capítulo 3: Algoritmo propuesto para la generación de curvas paralelas en el guiado de tractores

En este capítulo se presenta de forma general un algoritmo para la generación de curvas paralelas. Posteriormente se presentará la problemática derivada de este, así como una mejora para evitarla. La mejora consiste en diseñar un algoritmo de enderezado de curvas.

3.1 Algoritmo inicial propuesto

El algoritmo inicial utilizado para la generación de curvas paralelas se basa en método matemático de generación a partir del vector normal. En casos reales no es común que la curva a paralelizar este definida mediante una función o de forma paramétrica. Lo que se tiene son una serie de pares de puntos que se corresponden a las coordenadas *x* e *y* de la curva. Esto requiere adaptar el método de generación de paralelas.

Para cada par de puntos de la curva se calculan las coordenadas *x* e *y* del vector que los une. Esto se consigue restándole al segundo punto el primero. Este vector equivale al tangente en una curva definida mediante una función. Entonces, queda calcular la normal de este intervalo, que se consigue girando 90º el vector, es decir, se intercambian las coordenadas y poner una de ellas negativa(4), y hacerlo unitario.

$$(x, y) \rightarrow (-y, x) \qquad (4)$$

Por último, el vector normal unitario calculado se multiplica por la distancia a la que se desea que se encuentre la paralela y se suma al primer punto del intervalo para obtener las coordenadas del punto desplazado para generar la paralela. Se realiza este proceso para todos los pares de puntos, es decir, en lugar de existir una única expresión para el vector normal como en el apartado 2.2, se calculan tantos vectores normales como intervalos tenga la curva. En la **Figura 3.1** se puede ver cómo se aplicaría este algoritmo para tres intervalos de la curva.

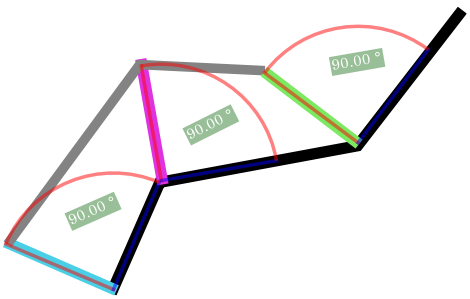

Figura 3.1. Paralelización a partir de vector normal. Los intervalos de la curva original están en negro. Los vectores normales de cada uno de los intervalos son de color cian, magenta y lima. Los intervalos en gris son los paralelos.

3.2 Problemática del anterior algoritmo

La generación de curvas paralelas tiene un problema principal. Cuando se realizan sucesivas paralelizaciones o estas se realizan con mucha separación aparecen discontinuidades que tienen el nombre de cúspides [8]. La curva azul se corresponde con la trayectoria original, las siguientes curvas en magenta son las paralelas. En la octava paralela aparece una cúspide. **Figura 3.2.** Estas cúspides aparecen en el lado cóncavo de la curva cuando se reduce la curvatura.

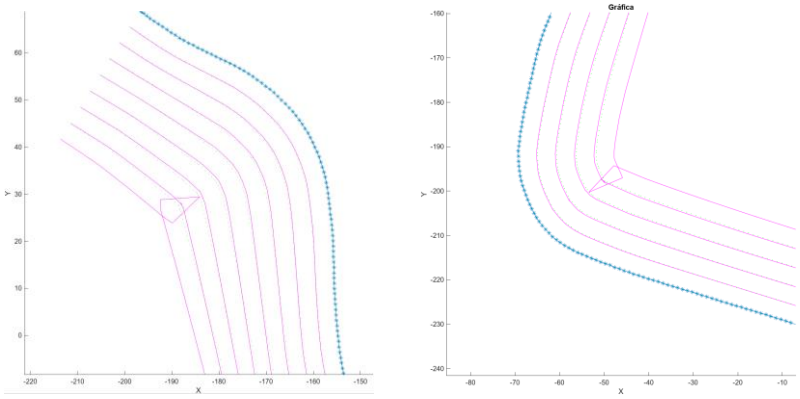

Figura 3.2. Se ilustra la formación de una cúspide en dos curvas paralelas. Las cúspides aparecen cuando la curvatura disminuye demasiado debido al proceso de generación de paralelas. Izquierda: La curva azul se corresponde con la trayectoria original, las curvas magenta son las paralelas. Aparece una cúspide en la octava curva. Derecha: La curva azul se corresponde con la trayectoria original, las curvas magenta son las paralelas. Aparece una cúspide en la quinta curva

A la hora de trasladar este problema al mundo real aparece el principal inconveniente. Un tractor no puede cambiar de dirección en el sitio, necesita una curva continua por lo que es esencial evitar que aparezca este fenómeno.

Para ello se ha ideado un algoritmo de enderezado teórico que combinado con un algoritmo de paralelización reduce en gran medida la aparición de este fenómeno como se verá en posteriores apartados.

3.3 Mejora del algoritmo anterior: Enderezamiento de trayectorias[1]

En este subapartado se propone un algoritmo de enderezado que se combinará con el de generación de paralelas para mejorarlo evitando la aparición de cúspides.

El modelo de enderezamiento que se ha ideado para este trabajo consiste en intentar alcanzar el recorrido de mínima distancia teniendo en cuenta el desplazamiento respecto a la posición original. El recorrido de mínima distancia siempre será la recta que una los dos extremos de la curva. En la **Figura 3.3** está ilustrado como se espera que degenere la curva en una recta tras la aplicación del algoritmo. La curva roja es la curva original y las otras curvas azules son el efecto deseado del algoritmo sobre el seno que es la eventual reducción de la curvatura para reducir el problema mencionado en el subapartado anterior.

---

[1] Agradecer al Dr. Cesar Palencia de Lara por su ayuda en la creación del algoritmo de enderezado.

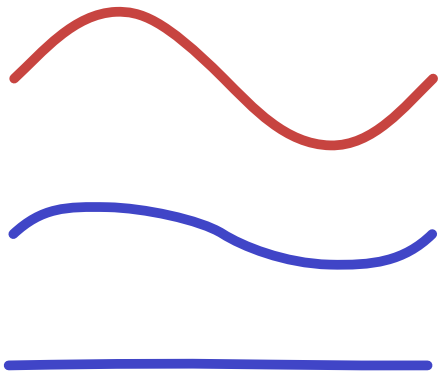

Figura 3.3. Se muestra el efecto esperado de enderezado en un seno. La curva original, en rojo, se endereza a través de varias iteraciones, produciendo una curva azul con una menor curvatura, evitando la formación de cúspides y finalmente eliminando completamente la curvatura como se ve en la última curva azul.

Para restringir el enderezado y evitar que degenere inmediatamente en una recta se añadirán unos límites superior e inferior que acoten la desviación máxima que puede tener la curva. Estos límites serán referidos como Solape máximo (SM) y Sin tratar Máximo (STM) debido a que estamos trabajando en un caso de uso agrícola. Suponiendo que la curva paralela a generar tras el enderezado se encuentra debajo, SM es la longitud de la curva amarilla y STM es la longitud de la curva verde en la **Figura 3.4**.

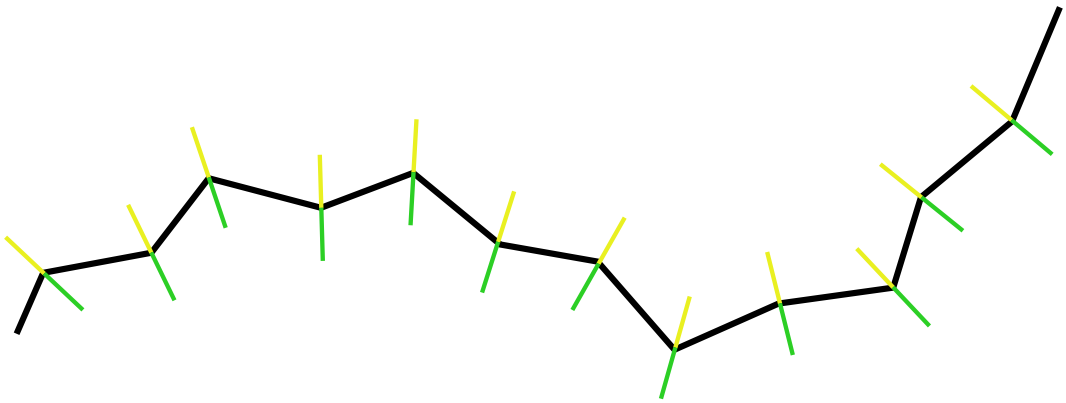

Figura 3.4. Se representan dos parámetros clave para cada punto de la curva. El Solape máximo (SM), en amarillo, y el Sin tratar máximo (STM), en verde. La curva original aparece en negro. Definen la máxima distancia que puede ser modificada la curva original en el enderezado.

Para el desplazamiento de los puntos entre SM y STM se creará la función de la **Figura 4.11** de tal forma que, para cada punto, exista un coste de desplazamiento respecto a su posición original. Este coste se incrementa asintóticamente hasta los límites definidos anteriormente. Es posible utilizar otro tipo de funciones mientras estas sean asintóticas. Entonces, conseguimos un balance según la fórmula (5). Cada Mi es un valor que deriva de una función arbitraria pero que debe ser asintótica (la función elegida se especifica en el apartado 4.2.4 y la R corresponde a la distancia recorrida por el tractor. T es una cantidad derivada del balance anteriormente mencionado. El objetivo es minimizar T, es decir, se busca minimizar el recorrido teniendo en cuenta una penalización por desviación respecto a la curva original.

$$T = \sum M_i + R \qquad (5)$$

De esta forma se puede definir el enderezamiento de la curva como un problema de optimización del que se puede diseñar un método para encontrar la solución.

El método de diseño que se propone en este trabajo consiste en una solución por fuerza bruta. Se realiza de forma iterativa como se explica en los siguientes pasos.

Los desplazamientos SM y STM se encuentran en la dirección de la bisectriz del ángulo que forman las dos rectas a izquierda y derecha del punto, por ejemplo, para un ángulo recto sería como se ve en la **Figura 3.5.** La elección de esta dirección se ha decidido de forma arbitraria con base en que es la línea que divide el ángulo en dos partes iguales.

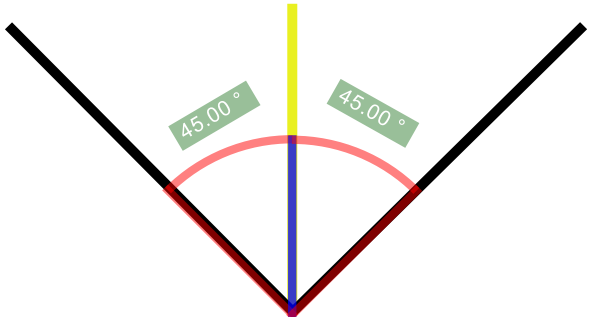

Figura 3.5. Esquema de la bisectriz de un ángulo recto. La bisectriz, que divide el ángulo en dos partes iguales, se emplea como dirección en la que generar los desplazamientos en el enderezado de las curvas paralelas.

Posteriormente, se calcula la distancia inicial del recorrido de la trayectoria (T=R). A continuación, para cada punto independientemente, se comprueba si alguno de los desplazamientos generados en la bisectriz reduce T según la fórmula indicada anteriormente (5)**.** Esto puede ocurrir porque, a pesar de que al desplazar el punto aumenta *Mi*, la distancia del recorrido de la trayectoria se puede reducir lo suficiente para compensar y superar ese aumento. Es importante destacar que cada punto que se comprueba se hace respecto a la curva original, es decir, no se modifica la R utilizada para cada uno de los puntos de la curva. En la **Figura 3.6**, se puede ver un esquema de los desplazamientos para cada punto de la curva original. Los posibles desplazamientos del punto original se encuentran equidistantes entre SM y STM. El punto con mayor grosor en el esquema es el que se debería elegir en cada caso. Una vez recorridos todos los puntos, aplico los desplazamientos que han resultado mejorar T obteniendo una curva que se intuye que va a reducir ligeramente la curvatura.

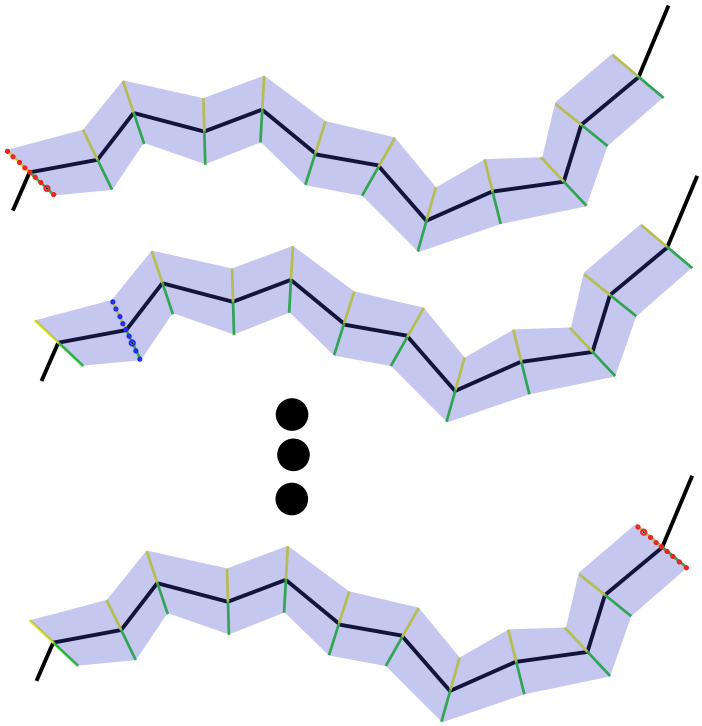

Figura 3.6. Desplazamientos generados entre los límites del "Solape máximo" y "Sin tratar máximo" para cada punto de la curva. Se calculan un número determinado de desplazamientos para cada punto y se elige el que mejora la curva. Se hace tantas veces como número de puntos haya. En estas tres curvas se muestran los desplazamientos para el primer punto, segundo y último como ejemplos.

Se espera conseguir la curva indicada en **Figura 3.7** tras la ejecución del algoritmo. La zona azul difuminada se corresponde al área en la que estará encerrada la nueva trayectoria. La curva en negro es la original y la roja es la enderezada tras el algoritmo. Hay que destacar que la figura es un esquema para ilustrar la situación esperada, la forma que adquiera el enderezado tras la implementación del algoritmo puede ser diferente.

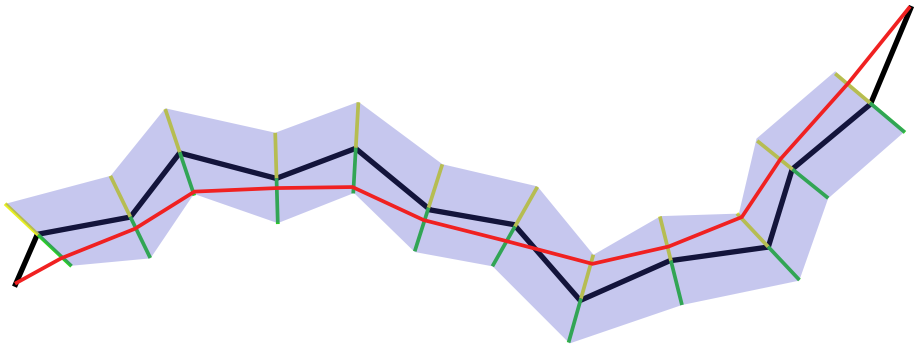

Figura 3.7. Se muestra un esquema de la trayectoria resultante después de aplicar el algoritmo de enderezado. La curva negra representa la trayectoria original, la curva roja se corresponde con la curva enderezada, con menor curvatura dentro del área azul definida por el Solape máximo (SM) y el Sin tratar máximo (STM).

Por último, queda repetir los pasos indicados en el párrafo anterior las veces que se considere oportuno tomando como referencia la curva generada en la iteración previa en lugar de la original. Con esto se completa el proceso de enderezado de la curva. Cada una de estas iteraciones se denominará subproceso de enderezado.

Estos dos algoritmos son combinados de forma que se intercala el algoritmo de enderezamiento con el de paralelización consiguiendo un efecto más natural y que se espera que se acabe eliminando la curvatura tras una cantidad de paralelizaciones tal y como se puede apreciar en la **Figura 3.8.** En el esquema se endereza con dos paralelas, pero en la práctica la cantidad de curvas paralelas que se realizarán antes de eliminar la curvatura dependerán de la velocidad de enderezamiento. Esta velocidad directamente relacionada con la cantidad de espacio SM y STM que se deja. Cuanto mayor SM y STM más se enderezará la curva en cada iteración junto con la influencia de otros parámetros que se especificarán en el apartado 4.3.

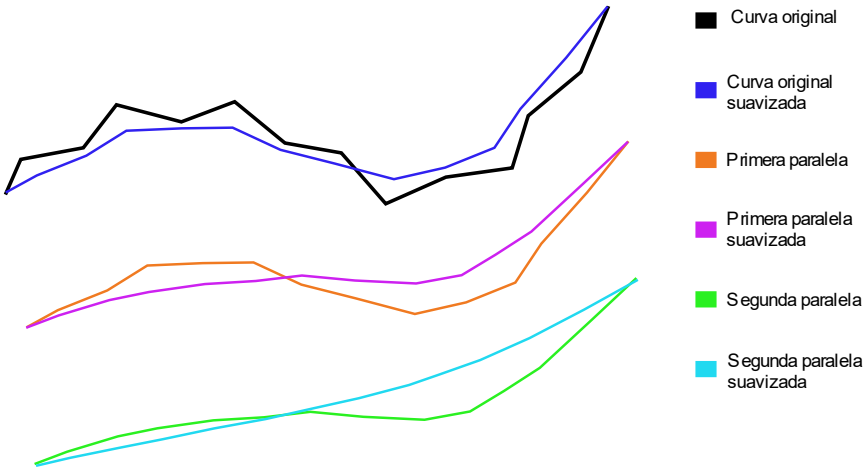

Figura 3.8. Efecto final de los algoritmos de enderezado y paralelización combinados. La curva enderezada en azul oscuro se genera a partir de la curva original en negro. A partir de esta curva azul se genera la paralela roja que se endereza en la morada y así sucesivamente hasta llegar a eliminar la curvatura.

3.4 Conclusiones

En este capítulo se ha presentado un algoritmo de enderezado. La necesidad del enderezado en las paralelas se presenta de forma concisa en el inicio de este capítulo. Se han ilustrado mediante esquemas los diferentes pasos que posee el algoritmo. También se ha expuesto como este algoritmo se combina con el de paralelización para obtener un mejor resultado. Este algoritmo junto con la paralelización mediante vector normal constituye la base de la aplicación diseñada en el siguiente apartado.

Capítulo 4: Aplicación desarrollada y resultados obtenidos

Capítulo 4: Aplicación desarrollada y resultados obtenidos

En este capítulo se describirá la aplicación desarrollada. La aplicación se ha desarrollado utilizando appDesigner, una extensión de MATLAB que permite crear aplicaciones web y de escritorio[9]. Mediante esta extensión he creado una interfaz gráfica de usuario que permite la modificación de parámetros de una forma más directa y práctica que mediante un script. Se puede ver una vista de la interfaz de forma general en la **Figura 4.1.**

Una vez abierto, se ve que el fichero MATLAB se divide en dos secciones. La distribución de la interfaz se desarrolla en el apartado 4.1 que corresponde a la sección de "Design view" y el código se describe en el apartado 4.2 que responde a la sección de "Code view". En el apartado 4.3 se presentan diferentes curvas paralelizadas mediante el algoritmo implementado, con variaciones de parámetros.

4.1 Descripción de la aplicación desarrollada

El algoritmo utilizado en esta aplicación se basa en la mejora del algoritmo descrita en el apartado 3.3**¡Error! No se encuentra el origen de la referencia.** con algunos cambios que apreciarán tanto en e ste subapartado como en el siguiente donde se detallarán las partes más importantes del código. A continuación, en este subapartado, se describirá la interfaz y sus elementos de forma más pormenorizada.

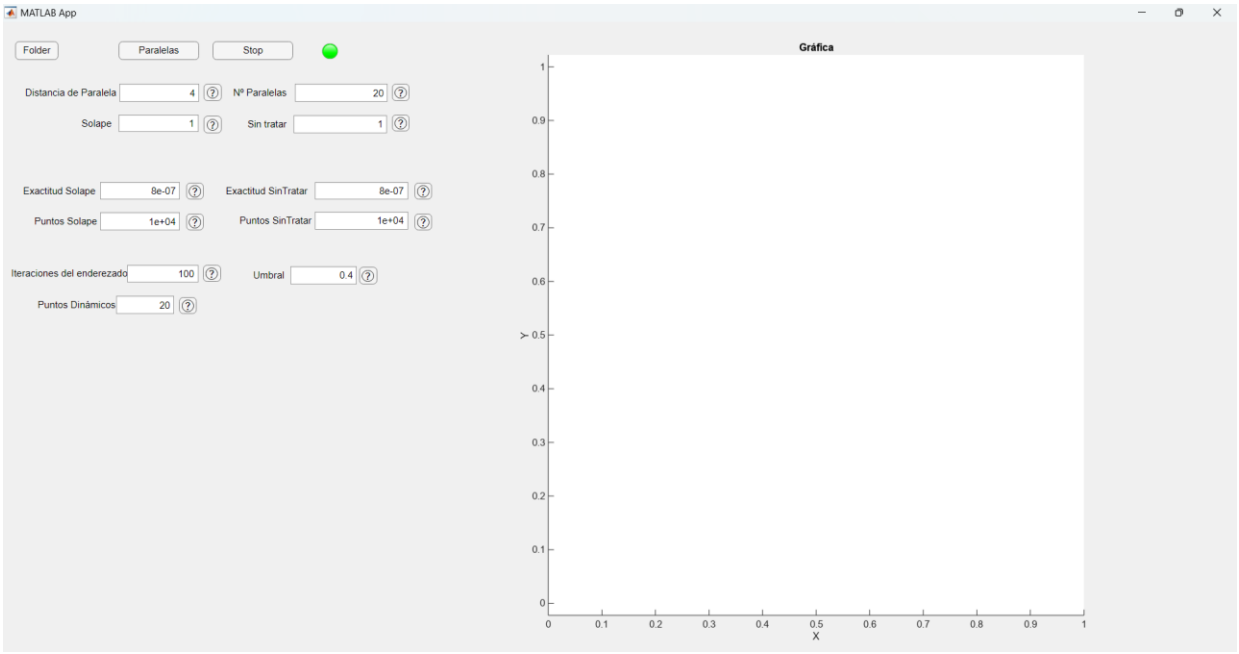

Figura 4.1. Interfaz gráfica de la aplicación desarrollada en MATLAB para la generación de curvas paralelas. La interfaz permite cargar curvas originales y generar las paralelas, mediante once parámetros configurables diferentes, mostrando todas estas curvas mediante una representación gráfica. También permite parar la ejecución en cualquier momento.

Lo primero que se aprecia de forma más destacada es la gráfica. Esta es el área de representación de curvas. Representa las curvas originales en azul, las curvas enderezadas mediante puntos verdes y las curvas paralelas en magenta.

En la zona superior izquierda se pueden ver tres botones y un icono verde. **Figura 4.2**.

El botón de nombre *Folder* es el encargado de cargar en la aplicación las curvas originales. Para ello, abre un menú en el sistema operativo desde donde puedes seleccionar el fichero deseado. Posteriormente representa la curva cargada en el área gráfica.

El botón *Paralelas* tiene implementada la función de enderezamiento y paralelización de las curvas. Esta función se ejecuta en función de diversos parámetros cuya utilidad se explicará a continuación en este apartado. Cada una de las curvas generadas, tanto la enderezada como la paralela, se representan en el área gráfica dinámicamente según se van generando. Al acabar la ejecución aparecerá un mensaje informativo como el de **Figura 4.3**.

El botón *Stop* realiza la función de detener la ejecución del enderezamiento y paralelización de forma prematura. De esta forma, si algo no sale como debería, no es necesario esperar a que acaben todas las iteraciones para modificar los parámetros e intentar otra ejecución.

Por último, hay que indicar que la lámpara con luz verde sirve como indicador de que la paralelización no se está ejecutando en ese instante y por lo tanto la aplicación esta libre. Mientras la luz está roja, se está ejecutando y es recomendable no modificar ningún parámetro ni tocar ningún botón excepto el de *stop* para evitar comportamientos no deseados.

Figura 4.2. Vista de los tres botones de la interfaz: Folder, Paralelas y Stop. Permiten cargar curvas, iniciar el proceso de paralelización y detener la ejecución respectivamente. También se observa una lámpara que indica el estado de ejecución, verde si la aplicación esta libre, rojo si se esta ejecutando la paralelización

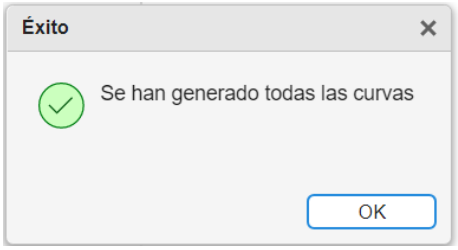

Figura 4.3. Mensaje de confirmación que aparece al finalizar exitosamente la generación de curvas paralelas. Indica que se han generado todas las curvas paralelas.

A continuación, se describirán los parámetros. Hay once parámetros distintos configurables.

Los primeros Cuatro que se ven debajo de los botones son los parámetros principales para caracterizar la generación de curvas paralelas. Estos parámetros se ven en la **Figura 4.4** . El parámetro *distancia paralela* como su propio nombre indica, se refiere a la distancia de cada curva paralela respecto a la anterior y, en primera instancia respecto a la original. Esta distancia también puede ser negativa si se desea que la paralelización se realice en la otra dirección. En ese caso, los nombres de *Solape* y *Sin tratar* se intercambiarán de posición por implementación. *Solape* indica el solape máximo en metros (SM) que permite el usuario. *Sin tratar* sigue el mismo concepto, pero refiriéndose al espacio que el usuario decida dejar sin tratar (STM). *N.º Paralelas* indica el número de paralelizaciones que se van a realizar, es decir, el número de curvas paralelas que habrá al final de la ejecución.

Posteriormente, en la **Figura 4.5**, vemos los otros dos parámetros que afectan al Solape. La modificación de estos parámetros, especialmente el número de puntos puede afectar al rendimiento de la aplicación. *Exactitud Solape* es un coeficiente de la función M (consultar apartado 4.2.4) que indica la velocidad de enderezado en la dirección del solape, cuanto menor sea, mayor será el enderezado en cada iteración. *Puntos Solape* indica el número de puntos de desplazamiento que se crean entre la curva original y el máximo solape permitido. Los parámetros de sin tratar siguen las mismas ideas.

A continuación, en la **Figura 4.6**, se ven los últimos tres parámetros. El primero que aparece es *iteraciones del enderezado*. Este parámetro indica el número de veces que se va a repetir el subproceso de enderezado (ver apartado **¡Error! No se encuentra el origen de la referencia.**) antes de realizar la p aralelización. El parámetro *umbral* marca una distancia mínima entre puntos, de forma que si los puntos están demasiado juntos entre sí se eliminan. Esto se realiza para evitar la superposición entre puntos o incluso la inversión en orden de estos. El parámetro *Puntos Dinámicos* indica la cantidad de puntos de desplazamiento que se escogen en cada iteración del enderezado. Para cada subproceso del enderezado no se trabaja con todos los puntos de desplazamiento ya que daría serios problemas de rendimiento. En su lugar se escoge un número de puntos significativamente menor para trabajar, del orden de las decenas suele ser suficiente. En la memoria de programación se explicará la implementación de este mecanismo de forma más detallada.

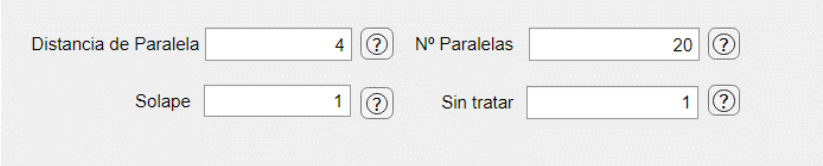

Figura 4.4. Parámetros principales para la generación de curvas paralelas en la aplicación. Se configura la distancia de paralela, el número de paralelas a generar y los espacios de Solape máximo y Sin tratar máximo que se permitirán dejar.

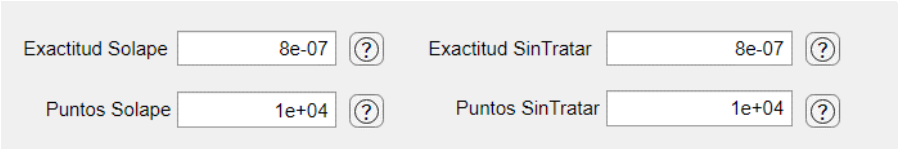

Figura 4.5. Configuración adicional de los parámetros Solape y Sin tratar. Exactitud Solape sigue una relación inversa con la velocidad de enderezado de la curva de forma que cuanto más pequeño sea este valor más rápido se desplaza hacia solape máximo. Puntos Solape indica el número de desplazamientos que se generan entre la curva y el solape máximo.

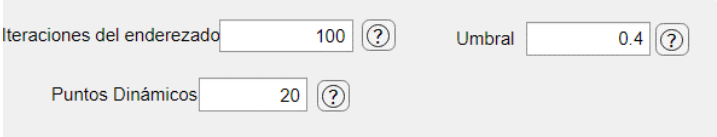

Figura 4.6. Muestra el número de iteraciones del proceso de enderezado, el umbral para eliminar puntos demasiado cercanos y el número de puntos dinámicos utilizados dentro de cada iteración del enderezado.

Los interrogantes que hay junto a cada uno de los parámetros son botones de ayuda. Al pulsarlos aparece una breve descripción del parámetro correspondiente. Esta descripción aparece en una ventana informativa tal y como la **Figura 4.7**.

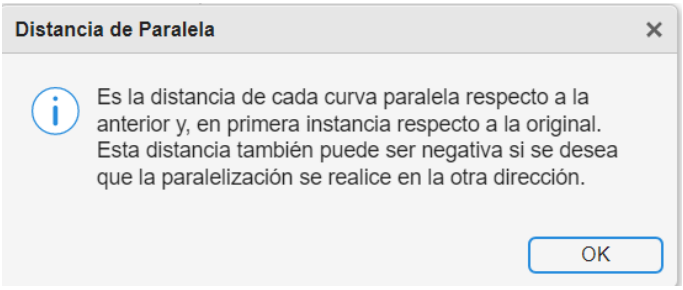

Figura 4.7. Ejemplo de un cuadro informativo que aparece al pulsar los botones de ayuda en la interfaz. Proporcionan una breve descripción de cada uno de los parámetros.

4.2 Memoria de Programación

El código Matlab sigue una estructura general de forma que las propiedades se encuentran lo primero. Después, está la parte principal del código donde se encuentran todas las funciones callback que interactúan directamente con la interfaz de la aplicación. Al final, se encuentran en gris más oscuro, las funciones que se encargan de crear los diferentes componentes de la aplicación y las encargadas de construir y borrar la aplicación.

4.2.1 Propiedades

En esta aplicación se usan tres propiedades globales. *t* y *curva* son las encargadas de almacenar las coordenadas x e y de la curva original. Durante la ejecución del algoritmo de paralelización y enderezado pasa a almacenar las coordenadas de la curva correspondientes a la iteración del enderezado en la que se encuentra la aplicación o la curva paralela generada si ya ha terminado el enderezado.

La propiedad *pararEjec* tiene valor cero o uno y sirve como identificador para indicar si la aplicación debe seguir ejecutándose o no.

4.2.2 startupFcn

Esta función se ejecuta al abrir la aplicación y tiene como objetivo ajustar algunos parámetros iniciales. **Figura 4.8** . Pone la aplicación en pantalla completa y ajusta los ejes en formato *equal* para que estos tengan el mismo tamaño para las coordenadas x e y. También configura la lámpara para que tenga inicialmente luz verde y así indicar que la aplicación está preparada.

```
function startupFcn(app)
    app.UIFigure.WindowState = 'maximized';
    axis(app.UIAxes, 'equal');
    app.Lamp.Color='g';
end
```

Figura 4.8. La función startupFcn Se ejecuta al abrir la aplicación. Ajusta la maximización de la pantalla, la visualización de los ejes y el estado de la lámpara indicadora.

4.2.3 FolderButtonPushed

La función de la **Figura 4.9** es la encargada de cargar y representar curvas desde ficheros *.txt* en la aplicación.

Con *uigetfile* se abre un menú del explorador de archivos que te permite escoger el fichero de texto a cargar. Después, se construye la ruta completa al fichero mediante *fullfile*. El fichero de texto deberá tener un formato tal que cada punto deberá estar en una línea diferente y las coordenadas x e y deberán estar separadas por el signo punto y coma (*;*). Mediante *readlines* la aplicación obtiene un vector en el que cada uno de los elementos se corresponde a una línea del fichero de texto. Posteriormente, utilizando *unique*, se eliminan del vector todos los puntos repetidos, es decir, aquellos elementos que tengan ambas coordenadas x e y iguales a las de otro anterior. A continuación, con *replace*, se sustituyen todas las comas y se colocan en su lugar puntos, ya que MATLAB utiliza el punto decimal. El siguiente paso consiste en separar cada uno de los elementos del vector en sus coordenadas x e y, usando *split*, así como pasarlo a formato numérico, usando *str2double*. Se cambia de signo la coordenada x para reorientar la curva además de quedarse con uno de cada tres puntos por cuestiones de rendimiento. Finalmente se representa la curva en el área gráfica.

Todo este código anteriormente mencionado se encuentra dentro de un bloque *try/catch*. Este bloque sirve como manejo de errores, de forma que si existiese algún problema a la hora de cargar el fichero de bloques la aplicación no se cerraría y en su lugar aparece un mensaje de error.

```
        function FolderButtonPushed(app, event)
              [A,path] = uigetfile(".txt");
              try
                fullpath=fullfile(path,A);
                curvasTxt= readlines(fullpath);
                curvasTxt=unique(curvasTxt,'stable');
                curvasPuntos=replace(curvasTxt,",",".");
                curvas=str2double(split(curvasPuntos(2:end-1),";"));
                curvas(:,1)=-curvas(:,1); %Para reorientar la curva
                app.t=curvas(1:3:end,1);
                app.curva=curvas(1:3:end,2);
                plot(app.UIAxes, app.t,app.curva, Marker="*")
              catch
                uialert(app.UIFigure,'No se ha podido cargar la curva', ...
                    "Error Message");
              end

        end
```

Figura 4.9. La función FolderButtonPushed se encarga de cargar curvas desde ficheros .txt y representarlas en el área gráfica de la aplicación. En caso de que el fichero a cargar no sea válido se mostrará un cuadro de alerta.

4.2.4 ParalelasButtonPushed

Es la función principal de la aplicación, dentro está el código que se encarga de la paralelización y del enderezado.

Lo primero que se hace es cambiar el color de la lámpara a rojo para indicar que la función se está ejecutando y configurar el valor de la propiedad global *pararEjec* a cero, esta propiedad hace que se pueda detener la ejecución cuando el valor cambie a uno, como se verá más adelante.

Posteriormente, se obtendrán de la interfaz los parámetros que se usarán en la función M. En concreto se obtendrán *Solape* y *ExactitudSolape*, así como *SinTratar* y *ExactitudSinTratar*. También se inicializará la función M tal y como se ve en la **Figura 4.10**. Es una función a trozos siendo (6) el trozo para t>0 y (7) el trozo para t<=0. Donde *t* es la variable a la que posteriormente se le especificará su valor, *c* y *d* son los valores de *ExactitudSolape* y *ExactitudSinTratar* respectivamente y *a* y *b* son el inverso de *Solape* y de *SinTratar* respectivamente. Esto es para que la asíntota de esta función tienda al valor que introduzca el usuario. Por ejemplo, si el usuario introduce el valor dos en *Solape* y *SinTratar*, *a* será igual a un medio y *b* será igual a un medio. Suponiendo que *c* y *d* valiesen uno la gráfica tendría la siguiente forma **Figura 4.11**. En la práctica se utilizarán *c* y *d* más pequeñas para darle más importancia a la reducción del recorrido cuando se modifique un punto. Con *c* y *d* del orden de 10^{-4} obtenemos la gráfica **Figura 4.12** con una pendiente mucho menor que propicia que M influya de forma menor en el balance de la fórmula (5) pero evitando que el recorrido reducido se salga de los límites del *Solape* y *SinTratar*.

```
SM=app.SolapeEditField.Value;
STM=app.SintratarEditField.Value;

a=1/SM;
b=1/STM;
c=app.ExactitudSolapeEditField.Value;
d=app.ExactitudSinTratarEditField.Value;

M= @(t) (t>0).*c.*(t.^2./(1-a*t)) + (t<=0).*d.*(t.^2./(1+b*t));
nPoints = size(app.t, 1)-2;
```

Figura 4.10. Configuración de la función M utilizada en el enderezado de curvas. c y d se corresponden con los valores de ExactitudSolape y ExactitudSinTratar respectivamente y a y b se corresponden el inverso de Solape y de SinTratar respectivamente. Es una función con dos asíntotas que tienden a Solape y Sin tratar.

$$M = c \cdot \frac{t^2}{1 - a \cdot t} \tag{6}$$

$$M = d \cdot \frac{t^2}{1 + b \cdot t} \tag{7}$$

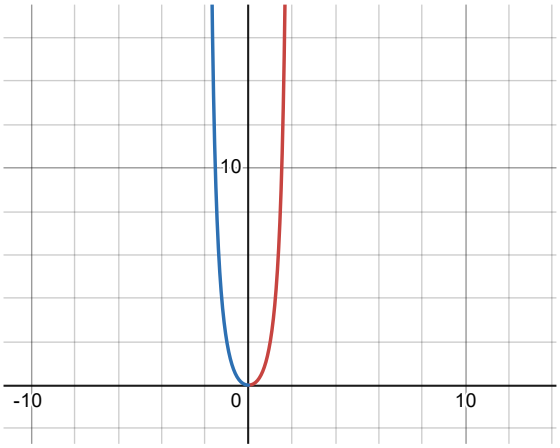

Figura 4.11. Representación gráfica de la función M con parámetros a = 1/2, b = 1/2, c = 1 y d = 1. La función posee asíntotas en x=2 y x=-2 que se corresponden al solape máximo y sin tratar máximo [10]

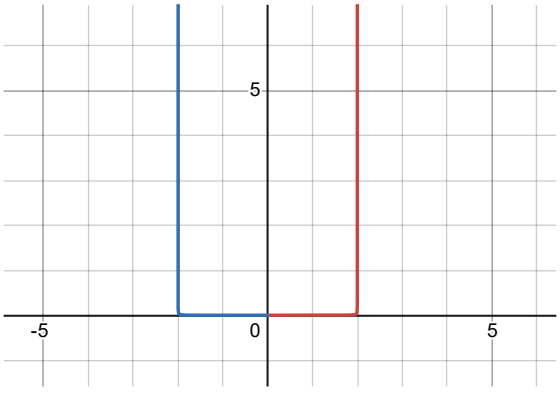

Figura 4.12. Gráfica de la función M con parámetros ajustados a valores mínimos: a = 1/2, b = 1/2, c = 0.0001 y d = 0.0001. Este ajuste permite obtener un menor efecto del enderezado en la generación de curvas paralelas [11]

Antes de entrar en el bucle solo queda crear una variable llamada *nPoints* que guardará el número total de puntos de la curva cargada menos dos, el inicial y el final. Con esta variable se creará un vector *ind* que almacenará los índices correspondientes al desplazamiento en ese momento de cada punto, es decir, un número de fila por cada columna de la matriz de puntos desplazados *tDesp*. Cada uno de los elementos de este vector se inicializará con el valor del parámetro *PuntosSinTratar*, por ejemplo, diez mil.

A continuación, se puede ver el inicio del bucle de ejecución de las curvas paralelas. Dentro de este bucle se encuentra prácticamente el resto del código de esta función. Este bucle se ejecutará mediante un *for* tantas veces como indique el parámetro *NParalelas* tal y como se ve en **Figura 4.13**.

```
for l=1:app.NParalelasEditField.Value
```

Figura 4.13. Inicio del bucle de iteraciones para la generación de paralelas. El número de veces que se recorre este bucle está definido por el parámetro número de paralelas.

Dentro del bucle la primera acción que se realiza es eliminar todos los puntos de la curva que no cumplan la condición del umbral. Para ello, tal y como se ve en la **Figura 4.14**, primero se calcula la distancia euclídea para cada par de puntos consecutivos en el vector de coordenadas *app.t* y *app.curva* que se corresponden a las coordenadas x e y en un plano. También se almacena en una variable el parámetro *Umbral*. Posteriormente, se recorre mediante un bucle *while* el vector de distancias generado comprobando en cada iteración si la distancia calculada es menor al umbral. En caso de que lo sea, se elimina el segundo punto del par, así como el punto correspondiente en el vector de índices de los puntos desplazados para que no haya incompatibilidades con los tamaños de los vectores más adelante. Por este mismo motivo hay que eliminar el elemento correspondiente del propio vector distancia y recalcular la nueva distancia entre dos puntos que ahora serán consecutivos al eliminar su intermedio. Esa nueva distancia ocupará el lugar e índice de la eliminada, es decir, todos los elementos se desplazarán una posición. Entonces, se volverá al inicio del bucle sin incrementar el contador y se comprobará si la nueva distancia cumple el umbral. En caso de que no se elimine ningún punto porque se cumple el umbral, el contador avanzará a la siguiente posición. Este bucle se realiza para todos los elementos del vector distancia o lo que es lo mismo, todos los pares de puntos.

```
        dist = sqrt((app.t(2:end) - app.t(1:end-1)).^2 + ...
            (app.curva(2:end) - app.curva(1:end-1)).^2);
        umbral = app.UmbralEditField.Value;
        % Eliminar puntos que están demasiado cerca
        i = 1; % Inicio desde el primer punto
        while i < length(dist)
            if dist(i) < umbral
                % Eliminar el siguiente punto en la secuencia
                app.t(i+1) = [];
                app.curva(i+1) = [];
                ind(i) = [];

                % Actualizar las distancias
                dist(i) = [];  % Eliminar la distancia
                dist(i) = sqrt((app.t(i+1) - app.t(i)).^2 + ...
                    (app.curva(i+1) - app.curva(i)).^2);

            else
                i = i + 1; % Avanzar al siguiente punto
            end
        end
```

Figura 4.14. Proceso de eliminación de puntos cercanos en la curva. La eliminación se realiza si la distancia entre puntos consecutivos es menor al umbral configurado, evitando la superposición y el cruce entre puntos.

El siguiente paso consiste en calcular la dirección sobre la que van a ir los puntos desplazados. Con tal fin se ha implementado el código que aparece en la **Figura 4.15**.

Primero se inicializa la variable *vect* con número de filas igual al número de puntos de *app.t* menos uno y dos columnas. En esta variable se almacenan vectores, la primera columna para la coordenada *app.t* y la segunda para *app.curva* que en este algoritmo se corresponden a las coordenadas cartesianas x e y. Se guardan los vectores diferencia entre todos los puntos consecutivos, en cada fila un vector, por eso se inicializó el vector con una fila menos que el número de puntos. A continuación, es necesario normalizar los vectores calculados. Con ese propósito se calcula la norma euclídea de cada uno de los vectores, es decir, de cada una de las filas de la matriz *vect*, y se almacena en la variable *Mag*. Inmediatamente después, para conseguir la normalización, se divide cada una de las filas de la matriz *vect* por su correspondiente elemento en el vector *Mag*. Posteriormente, se obtiene el vector suma de dos vectores consecutivos y este también se normaliza de la misma forma que se ha explicado anteriormente. El último paso consiste en girarlo 90 grados para que, más adelante, al situarlo en el punto intermedio común a los dos vectores consecutivos, este tenga la dirección de la bisectriz sobre la que se situarán los puntos desplazados.

```
vect=zeros(size(app.t,1)-1,2);

vect(1:length(app.curva)-1,2)=app.curva(2:end)-app.curva(1:end-1);
vect(1:length(app.curva)-1,1)=app.t(2:end)-app.t(1:end-1);

Mag=sqrt(sum(vect.^2, 2));
vectunit=vect./Mag;

vSum=vectunit(2:end,:)+vectunit(1:end-1,:);
Magbsic=sqrt(sum(vSum.^2, 2));
vSumnorm=vSum./Magbsic; %normalizo
vBsicnormgir=[-vSumnorm(:,2) vSumnorm(:,1)];
```

Figura 4.15. Cálculo de la bisectriz para cada par de puntos consecutivos en la curva. La bisectriz representa la dirección en la que se generan los puntos desplazados.

Una vez calculada la dirección de los puntos desplazados es hora de determinar la posición concreta de cada uno de estos puntos, el código aparece en **Figura 4.16**. Los puntos desplazados deben estar situados entre los valores de solape máximo (SM) y de Sin Tratar Máximo. Por ello se empieza creando dos vectores *kSM* y *kSTM* que almacenarán valores equiespaciados entre 0 y *SM* y entre -*STM* y 0. La separación entre estos valores dependerá del número de puntos que haya introducido el usuario a través de los parámetros *PuntosSolape* y *PuntosSinTratar*. Estos valores son usados para definir la posición de los puntos.

Para los desplazamientos hacia el solape máximo, se multiplica el vector dirección de los puntos desplazados por todos de los valores de *kSM* menos el correspondiente al cero. La multiplicación se separa en coordenadas, es decir, primero se calcula la nueva coordenada *x* y posteriormente la coordenada *y*. Esto se hace para todos los vectores, es decir, para todas las filas de la matriz *vBsicnormgir*. Si a las matrices resultantes de estas operaciones, es decir, una matriz para la coordenada *x* y otra para la y, se le suman todos los puntos menos el primero y el último se obtienen dos matrices, que almaceno en la variable *tDespPos* para la coordenada *x* y *curvaDespPos* para la coordenada *y*, en la que cada columna se corresponde a los desplazamientos de cada coordenada hasta solape máximo, con el número de puntos desplazados que se indicó en el parámetro *PuntosSolape*. Para el desplazamiento hacia sin tratar máximo, la principal diferencia radica en que al realizar la multiplicación entre la cada columna del vector *vBsicnormgir* y el *kSTM* en esta ocasión sí se tiene en cuenta el cero. De este segundo proceso similar obtengo también dos matrices que almaceno en *tDespNeg* y *curvaDespNeg*

Posteriormente, concateno las matrices *tDespPos* y *tDespNeg* para poder operar con una sola matriz en la que cada columna se corresponde a todos los desplazamientos desde Sin tratar máximo hasta solape máximo. Se hace lo mismo para *curvaDespPos* y *curvaDespNeg*.

Antes de entrar en el bucle de enderezado queda un último paso que se corresponde a calcular la penalización para cada desplazamiento. Esta penalización se almacenará en una variable de nombre *MM*. Se calcula esencialmente llamando a la función M con argumento de entrada el vector equidistante *kSM* y posteriormente se concatena con otra llamada a la función M esta vez con argumento *kSTM*. Se obtiene un vector con todas las penalizaciones por desplazamiento. En este caso no es una matriz porque todas estas penalizaciones no dependen de si es el tercer punto de la curva o el antepenúltimo.

```
kSM=linspace(0,SM,app.PuntosSolapeEditField.Value);
kSTM=linspace(-STM,0,app.PuntosSinTratarEditField.Value);

tDespPos=app.t(2:end-1)+vBsicnormgir(:,1)*kSM(2:end);
curvaDespPos=app.curva(2:end-1)+vBsicnormgir(:,2)*kSM(2:end);

tDespNeg=app.t(2:end-1)+vBsicnormgir(:,1)*kSTM;
curvaDespNeg=app.curva(2:end-1)+vBsicnormgir(:,2)*kSTM;

tDesp=[tDespNeg tDespPos];
curvaDesp=[curvaDespNeg curvaDespPos];

MM=M(kSM(2:end));
MM=[M(kSTM) MM ];
```

Figura 4.16. Calculo de los puntos desplazados entre Solape máximo y Sin Tratar máximo. También se generan las penalizaciones por desplazamiento y se almacenan en la variable MM.

Después del proceso anterior, se inicia un bucle anidado al anterior, el bucle de enderezado. Este bucle se ejecutará mediante un *for* tantas veces como indique el parámetro *IteracionesdelEnderezado* tal y como se ve en **Figura 4.17**.

```
%número de iteraciones del algoritmo
nrep=app.IteracionesdelEnderezadoEditField.Value;
for k=1:nrep
```

Figura 4.17. Inicio del bucle para realizar el proceso de enderezado. El número de veces que se recorre este bucle está definido por el parámetro iteraciones del enderezado.

Dentro del bucle lo primero que se consulta es si la variable global *app.pararEjec* está con valor uno o cero. En caso de que esté con valor uno significará que se habrá pulsado el botón *stop* justo como se verá en el apartado 4.2.5. Entonces, se pondrá la lámpara de color verde y se detendrá la ejecución del bucle. Para ello se utiliza simplemente el siguiente código. **Figura 4.18**.

```
if app.pararEjec
    app.Lamp.Color='g';
    break;
end
```

Figura 4.18. Este código se ejecuta en caso de que la variable global app.pararEjec se ponga a uno, es decir, si se ha pulsado el botón de stop. Se configura la linterna con luz verde y se sale del bucle.

El siguiente paso comienza por calcular la distancia euclídea entre puntos consecutivos de la curva y almacenarlo en un vector *dist*. También se guarda en la variable *Patomar* el parámetro *PuntosDinmicos* que introduce el usuario. Por la implementación diseñada para esta tarea es necesario que *Patomar* sea número par. Para ello, como se puede ver en la **Figura 4.19**, se verifica la paridad comprobando si la variable es divisible entre dos. En caso de que no lo sea se genera una alerta por pantalla para informar al usuario de que el parámetro *PuntosDinmicos* introducido debe ser par. A continuación, se inicializa una matriz tridimensional llamada *distModif* y una matriz bidimensional llamada *dynamicIndex*.

La matriz *distModif* se crea para almacenar distancias. Para cada punto que pueda ser desplazado, es decir, todos menos el primero y el último, se calculará una matriz bidimensional en la que cada

columna se corresponderá al vector de distancias para cada desplazamiento del punto al que corresponde la matriz. Solo variarán los elementos del vector distancias que se vean afectados por el desplazamiento de los puntos en cada ocasión. Como se puede ver en la **Figura 4.19** solo se modifican dos filas por cada iteración del *for*. Esto es porque solo se desplaza un punto y, por lo tanto, solo varían dos elementos del vector distancia, es decir, dos elementos de la columna. El resto de las filas de la matriz *distModif* son copias de las secciones que no se modificarán del vector distancias original. Se crean estas copias utilizando la función *repmat*.

Respecto a lo comentado anteriormente, destacar que solo se toman *Patomar+1* desplazamientos para construir la matriz. Por esto, *distModif* tiene, para la primera dimensión el tamaño de todos los puntos menos uno o lo que es los mismo el tamaño del vector distancias calculado anteriormente. La segunda dimensión tiene un tamaño de *Patomar+1* limitando los desplazamientos que se calculan en cada iteración. La tercera dimensión tiene un tamaño de todos los puntos menos el primero y el último, es decir, todos los puntos que pueden ser desplazados.

Los desplazamientos que se toman en cada iteración del enderezado son siempre una cantidad Patomar+1, pero no son siempre los mismos. En este punto es donde entra la matriz *dynamicIndex*. Para cada punto, a partir del valor que este posea en el vector *ind*, se crean otros P*atomar/2* valores sumando uno al elemento de *ind* por cada incremento. También se crean otros *Patomar/2* valores restando. Este proceso te genera vectores fila con *Patomar+1* elementos por cada punto de la curva. La variable *dynamicIndex* se utiliza para guardar en cada fila los *Patomar+1* elementos. Hay tantas filas como puntos desplazables existan. La matriz *dynamicIndex* además de usarse para indexar los valores adecuados de *tDesp*, también será necesaria para reconfigurar el vector *ind* con los nuevos desplazamientos óptimos encontrados. Dependiendo de la posición en la que se encuentre el elemento correspondiente de *ind* se han implementado dos métodos extra para evitar que se asignen índices negativos o mayores que el tamaño total de *tDesp*.

```
dist=sqrt((app.t(2:end)-app.t(1:end-1)).^2+(app.curva(2:end)-app.curva(1:end-1)).^2);
%Ahora hago un bucle que recorra todos los puntos
Patomar=app.PuntosDinmicosEditField.Value;
if mod(Patomar, 2) == 0

    distModif = zeros(size(app.t,1)-1, Patomar+1, size(app.t,1)-2);
    dynamicIndex=zeros(size(app.t,1)-2,Patomar+1);

    for i=1:size(app.t,1)-2
        if(ind(i)<=(Patomar/2) )
             dynamicIndex(i,:)=(1:(Patomar+1));
        elseif(ind(i)>(size(tDesp,2)-Patomar/2))
             dynamicIndex(i,:)=(size(tDesp,2)-Patomar:size(tDesp,2));
        else
            dynamicIndex(i,:)=(ind(i)-fix(Patomar/2)):(ind(i)+fix(Patomar/2));
        end

        if(i~=1)
            distModif(1:i-1,:,i)=repmat(dist(1:i-1),1,Patomar+1);
        end
        distModif(i,:,i)=sqrt((tDesp(i,dynamicIndex(i,1:end))-app.t(i)).^2+ ...
            (curvaDesp(i,dynamicIndex(i,1:end))-app.curva(i)).^2);
        distModif(i+1,:,i)=sqrt((app.t(i+2)-tDesp(i,dynamicIndex(i,1:end))).^2+...
            (app.curva(i+2)-curvaDesp(i,dynamicIndex(i,1:end))).^2);
        if(i~=length(tDesp))
            distModif(i+2:length(app.t)-1,:,i)=repmat(dist(i+2:end),1,Patomar+1);
        end
    end
else
     uialert(app.UIFigure,'El número de puntos dinámicos debe ser par',...
         "Warning Message","Icon","warning");
    return
end
```

Figura 4.19. Generación de la matriz distModif, utilizada para almacenar las distancias entre puntos consecutivos tras el desplazamiento. Cada punto de la curva original que puede ser desplazado se corresponde con una submatriz bidimensional en la que cada columna se corresponde con el vector de distancias de la curva modificado en función del desplazamiento que corresponda para el punto en la propia columna.

Una vez creada la matriz *distModif* y calculados todos sus elementos ya se puede calcular T. Se utilizará el código de la **Figura 4.20**. La matriz *R* posee en cada elemento un recorrido diferente. Cada recorrido diferente deriva de los desplazamientos de los diferentes puntos. El recorrido es la suma de cada uno de los vectores distancia, es decir, se utiliza la orden *sum(distModif)* para la primera dimensión de la matriz. Según el centro de ayuda de MATLAB [12], la orden *sum* sobre un arreglo tridimensional opera en la primera dimensión. El tamaño de esta dimensión en la matriz resultante se vuelve uno mientras que las demás dimensiones se mantienen igual. Entonces la matriz *R* resultante tendrá en cada columna los recorridos con diferentes desplazamientos para cada elemento y toda la columna para un solo punto en concreto. Existen tantas columnas como puntos de la curva sean desplazables, es decir, todos menos el primer y el último punto.

Indexando el vector de penalizaciones *MM* con la matriz *dynamicIndex* obtengo una matriz bidimensional que tendrá en cada fila los recorridos con diferentes desplazamientos para cada elemento y toda la fila para un solo punto en concreto. Para que coincidan las dimensiones con *R* es necesario

transponer la matriz *dynamicM* intercambiando así filas por columnas. De esta forma ambas matrices se pueden sumar y el resultado es otra matriz de las mismas dimensiones que se guarda en la variable *T*.

Por último, tal y como se explicó en el apartado **¡Error! No se encuentra el origen de la r eferencia.**, se busca minimizar el balance entre R y M. Para ello, utilizamos la función *min*. Aplicada sobre la matriz bidimensional *T,* el resultado es un vector fila *tempInd* en el que en cada elemento se almacena el índice del balance que ha sido mínimo. Este índice se corresponde con el del punto desplazado que ha generado ese balance. Técnicamente, los índices que se consiguen son relativos al número de columnas de la matriz *dynamicIndex*, es decir, si *Patomar* es veinte, el valor de los índices resultantes estará entre uno y veintiuno. Para crear la nueva curva será necesario indexar mediante índices generales, que tengan todos los puntos, las matrices *tDesp* y *curvaDesp*. Entonces, para pasar a los generales que se almacenan en el vector *ind*, se utiliza el código de la **Figura 4.21**.

```
R=zeros(size(distModif,2),size(distModif,3));
T=R;

R(:,:)=sum(distModif);
%Cada una de las columnas suma un MM diferente
dynamicM=MM(dynamicIndex)';
T(:,:)=R+dynamicM;

%Hacer reconversión de índices (1,4,5) a (204,194...)
[~,tempInd]=min(T);
```

Figura 4.20. Cálculo de la matriz T. Representa el balance entre la distancia total de la curva y la penalización de cada punto que se ha desplazado. El mínimo de esta matriz se corresponde con la curva de menor curvatura para los puntos dinámicos elegidos.

```
n = length(tempInd); % o el valor que necesites para n
ind = dynamicIndex((1:n) + (tempInd(1:n) - 1) * size(dynamicIndex, 1))
```

Figura 4.21. Conversión de los índices relativos de los desplazamientos elegidos para enderezar la curva a índices generales, lo que permite aplicar los cambios a todos los puntos de la curva. Los índices relativos aparecen por el uso de puntos dinámicos.

Por último, una vez calculados los índices, queda asignar los puntos de la nueva curva indexando *tDesp* y *curvaDesp* mediante el vector *ind* actualizado tal y como se menciona anteriormente. Esta nueva curva se asigna a las variables globales de *app.t* y *app.curva*. De esta forma se puede repetir el proceso de enderezado indefinidamente.

```
nuevat=app.t;
nuevaCurva=app.curva;
for i=1:size(app.curva,1)-2
    nuevat(i+1)=tDesp(i,ind(i));
    nuevaCurva(i+1)=curvaDesp(i,ind(i));
end

app.t=nuevat;
app.curva=nuevaCurva;
```

Figura 4.22. Creación de la nueva curva utilizando los índices de los desplazamientos que han sido obtenidos por la ejecución del algoritmo de enderezado previamente.

Fuera del bucle de enderezado, pero antes de llegar a la paralelización aparece el código de la **Figura 4.23**. Este código se utiliza para representar la curva enderezada tras todas las iteraciones anteriores. Esta curva se representa de color verde, signos de punto y con un grosor de punto de dos en la escala de la función *plot*.

```
hold(app.UIAxes,"on")
plot(app.UIAxes,nuevat,nuevaCurva,'Color','g','Marker','.',...
     'MarkerSize',2,'LineStyle','none');
drawnow;
```

Figura 4.23. Código para representar la curva enderezada tras la realización de todas las iteraciones del algoritmo de enderezado. Se representa mediante puntos de color verde en el área gráfica.

Una vez implementado el algoritmo de enderezado, queda hacer lo propio con el algoritmo de paralelización. Se ha utilizado el método de generación de paralelas a partir del vector normal. Este método se implementa con el código que aparece en la **Figura 4.24**. Primero se concatenan en una sola matriz *curvas* ambas coordenadas de la curva generada anteriormente. A continuación, se inicializan las variables *vect*, *vectUnit* y *normal* como matrices con un tamaño de dos columnas y filas iguales al número de puntos que tenga la curva a paralelizar. Son matrices porque cada columna se corresponde a una coordenada, la primera a *x* y la segunda a *y*. Una vez inicializadas las matrices, para cada par de puntos de la curva se calcula su diferencia que corresponde al vector entre los puntos, posteriormente se normaliza dividiendo el vector generado entre la norma que se calcula mediante la función *norm* y, por último, se gira noventa grados el vector y se pone en negativo una de las coordenadas para obtener la normal del vector entre los dos puntos. Las matrices generadas de cada operación se almacenan en *vect*, *vectUnit* y *normal* respectivamente.

Teniendo las direcciones normales para cada vector solo queda desplazar los puntos en esas direcciones la cantidad indicada por el usuario mediante el parámetro *DistanciaDeParalela*. Para ello se usan la tercera y cuarta línea de código empezando por el final de la **Figura 4.24**. Por implementación del algoritmo para el penúltimo y el último punto se utiliza la misma dirección normal. Esta nueva curva paralela se asigna a las variables globales de *app.t* y *app.curva*. De esta forma se puede repetir el proceso de generación de curvas paralelas indefinidamente.

```
curvas=[nuevat nuevaCurva];
vect=zeros(length(curvas)-1,2);
vectUnit=zeros(length(curvas)-1,2);
normal=zeros(length(curvas)-1,2);

for n=1:length(curvas)-1
    vect(n,:)=curvas(n,:)-curvas(n+1,:);
    vectUnit(n,:)=vect(n,:)./norm(vect(n,:));
    normal(n,:)=[-vectUnit(n,2) vectUnit(n,1)];

end
d= app.DistanciadeParalelaEditField.Value;

paralela=curvas(1:end-1,:)+normal(:,:)*d;
paralela(end+1,:)=curvas(end,:)+normal(end,:)*d;

app.t=paralela(:,1);
app.curva=paralela(:,2);
```

Figura 4.24. Implementación del algoritmo de paralelización utilizando el método del vector normal. A partir de la curva enderezada, se generan las curvas paralelas con una separación indicada por el parámetro distancia de paralela que configura el usuario.

Por último, antes de cerrar el bucle, se representa la curva paralela generada con las opciones por defecto de la función *plot* con el único cambio de que se configura el color como magenta.

```
plot(app.UIAxes,paralela(:,1),paralela(:,2),'Color','m');
drawnow;

hold(app.UIAxes,"off")
```

Figura 4.25. Código utilizado para representar la curva paralela. Se representa mediante una línea continua de color magenta

El último proceso que se realiza en esta función consiste en enviar un mensaje para informar al usuario que la ejecución de la función ha terminado. Además, se configura la luz de la lámpara de color verde indicando que la aplicación está disponible.

```
app.Lamp.Color='g';
uialert(app.UIFigure,'Se han generado todas las curvas',...
    "Éxito","Icon","success");
```

Figura 4.26. Mensaje de confirmación que se crea al finalizar la generación de curvas paralelas. Indica que se han generado todas las curvas paralelas deseadas.

4.2.5 StopButtonPushed

Esta función cambia la variable global *pararEjec* a uno. Como su nombre denota, se corresponde al *callback* del botón *stop*. De esta forma indica a la función principal *ParalelasButtonPushed* que debe detener su ejecución. Es una función muy simple y su código se puede ver en la **Figura 4.27**.

```
function StopButtonPushed(app, event)
    app.pararEjec=1;
end
```

Figura 4.27. La Función StopButtonPushed permite detener la ejecución de la función paralelasButtonPushed encargada de generar y representar las curvas paralelas. Para ello se configura la variable global app.pararEjec a uno

4.2.6 DistanciadeParalelaEditFieldValueChanged

Debido a la implementación de los parámetros *sin tratar* y *solape* en la función *ParalelasButtonPushed* es necesario que estos se intercambien de posición dependiendo de si la paralelización se hace en una dirección o en la contraria. Esta función se ejecuta cada vez que el parámetro *distancia de paralela* se modifica. Tal y como se ve en la **Figura 4.28**, primero se extrae el valor del parámetro mencionado y posteriormente se comprueba si es positivo o negativo. De acuerdo al resultado obtenido los títulos de los cuadros de numéricos de los parámetros *solape* y *sin tratar* se configuran de una forma y otra.

```
function DistanciadeParalelaEditFieldValueChanged(app, event)
    value = app.DistanciadeParalelaEditField.Value;
    if value < 0
        % Si la distancia es negativa, intercambiar los nombres
        app.SolapeEditFieldLabel.Text = 'Sin tratar';
        app.SintratarEditFieldLabel.Text = 'Solape';
    else
        % Si la distancia es positiva, restaurar los nombres
        app.SolapeEditFieldLabel.Text = 'Solape';
        app.SintratarEditFieldLabel.Text = 'Sin tratar';
    end
end
```

Figura 4.28. La función DistanciadeParalelaEditFieldValueChanged se encarga de intercambiar la posición de los parámetros Solape y Sin tratar en la interfaz en función de el signo que tenga el parámetro distancia de paralela. Esto se hace porque al cambiar la dirección de paralelización estos parámetros se intercambian en el código

4.2.7 Funciones de cuadros de información

En la interfaz, junto a cada uno de los parámetros a modificar se encuentran unos botones que al pulsarlos generan un evento *uialert* de tipo información (*info*) que muestra un mensaje aportando información sobre el parámetro en concreto. En la **Figura 4.29** se puede ver como ejemplo la función *callback*, es decir la que se ejecuta al pulsar el botón junto al parámetro *distancia de paralela*. El resto de los parámetros se han programado idénticamente cambiando el mensaje explicativo.

Los menajes que se muestran para cada parámetro son los siguientes:

Distancia de paralela: “Es la distancia de cada curva paralela respecto a la anterior y, en primera instancia respecto a la original. Esta distancia también puede ser negativa si se desea que la paralelización se realice en la otra dirección.”

Solape: “Indica el solape máximo en metros (SM), es decir, la máxima distancia que pueden desplazarse los puntos de la curva en el enderezado en la dirección de solape.”

Sin Tratar: “Indica el espacio sin tratar máximo en metros (SM), es decir, la máxima distancia que pueden desplazarse los puntos de la curva en el enderezado en la dirección sin tratar.”

Nº Paralelas: "Indica el número de paralelizaciones que se van a realizar, es decir, el número de curvas paralelas que habrá al final de la ejecución."

Exactitud Solape: "Es un coeficiente de la función M (consultar apartado 4.2.4 de la documentación) que indica la velocidad de enderezado en la dirección del solape, cuanto menor sea, mayor será el enderezado en cada iteración."

Exactitud Sin Tratar: "Es un coeficiente de la función M (consultar apartado 4.2.4 de la documentación) que indica la velocidad de enderezado en la dirección Sin tratar, cuanto menor sea, mayor será el enderezado en cada iteración."

Puntos Solape: "Indica el número de puntos de desplazamiento que se crean entre la curva original y el máximo solape permitido."

Puntos Sin Tratar: "Indica el número de puntos de desplazamiento que se crean entre la curva original y el espacio sin tratar máximo."

Iteraciones del Enderezado: "Indica el número de veces que se va a repetir el subproceso de enderezado (ver apartado 3.2 de la memoria) antes de realizar la paralelización."

Umbral: "Marca una distancia mínima entre puntos, de forma que si estos están demasiado juntos entre sí se eliminan. Esto se realiza para evitar la superposición entre puntos o incluso la inversión en orden de estos."

Puntos Dinámicos: "Indica la cantidad de puntos de desplazamiento que se escogen en cada iteración del enderezado. Para cada subproceso del enderezado no se trabaja con todos los puntos de desplazamiento ya que daría serios problemas de rendimiento. En su lugar se escoge un número de puntos significativamente menor para trabajar, del orden de las decenas suele ser suficiente."

```
function ButtonDistanciadeParalelaPushed(app, event)
     mensaje="Es la distancia de cada curva paralela respecto a " + ...
         "la anterior y, en primera instancia respecto a la original." + ...
         " Esta distancia también puede ser negativa si se desea" + ...
         " que la paralelización se realice en la otra dirección. ";
     uialert(app.UIFigure,mensaje,...
         "Distancia de Paralela","Icon","info");
 end
```

Figura 4.29. La Función ButtonDistanciadeParalelaPushed se ejecuta cuando se presiona el botón de ayuda junto al parámetro distancia de paralela y muestra un mensaje informativo sobre el propio parámetro. El mensaje se muestra mediante la función uialert.

4.3 Resultados de generación de curvas paralelas obtenidos

En este subapartado se presentarán los resultados del algoritmo, es decir, curvas paralelas obtenidas mediante el código anterior. Las curvas que se paralelizarán serán cuatro. Los puntos utilizados para crear las curvas han sido tomadas en una finca en Pozal de Gallinas. En la aplicación están configurados unos parámetros por defecto. Estos serán los parámetros utilizados en las diversas generaciones de curvas paralelas a no ser que se indique lo contrario y se especifiquen otros valores. Estos parámetros son:

- Distancia de paralela: 4
- N.º Paralelas: 20
- Solape: 1

- Sin tratar: 1
- Exactitud Solape: $8 \cdot 10^{-7}$
- Exactitud Sin tratar: $8 \cdot 10^{-7}$
- Puntos Solape: 10000
- Puntos Sin tratar: 10000
- Iteraciones del enderezado: 100
- Umbral: 0.4
- Puntos dinámicos: 20

4.3.1 Curva 1: Linde de la Parcela.

La primera curva presentada se corresponde a la linde de la parcela y se puede ver en la **Figura 4.30**. Solo tiene curvatura en una dirección y no es demasiado pronunciada.

Si generamos las curvas paralelas con los parámetros por defecto obtenemos las curvas que se ven en la **Figura 4.31**. Se ve que en la última curva paralela se ha reducido significativamente la paralela. Además, se puede apreciar el solape en las zonas donde las curvas paralelas se encuentran más juntas entre sí. Está acción es la que proporciona la eliminación de curvatura. Continuando la ejecución del algoritmo se puede llegar a eliminar por completo la curvatura. Si generamos diez paralelas más se puede ver que la curva se ha convertido en una línea recta, como se aprecia en la **Figura 4.32**. Aumentando el solape a 3 se puede conseguir un efecto parecido en 20 paralelas. **Figura 4.33**

También se puede paralelizar en la otra dirección. Para ello, se hace el parámetro de *Distancia de Paralela* negativo, es decir, se cambia a menos cuatro. Para veinte paralelas obtienes la **Figura 4.34**. En esta ocasión las paralelas se ven más separadas en algunas zonas. En este caso la acción de dejar espacio sin tratar es la que reduce la curvatura. Si se permite dejar más espacio sin tratar, es decir, aumentar el valor del parámetro *SinTratar* a tres, por ejemplo, consigo aproximarme a la recta en solo veinte paralelizaciones como se ve en la **Figura 4.35**

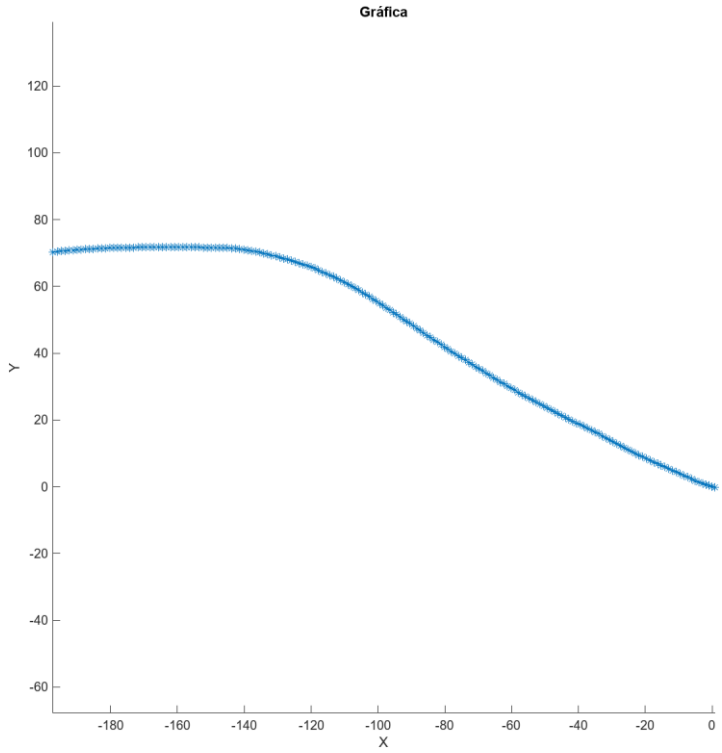

Figura 4.30. Linde de la parcela (Curva 1). Esta curva representa el contorno de una parcela agrícola.

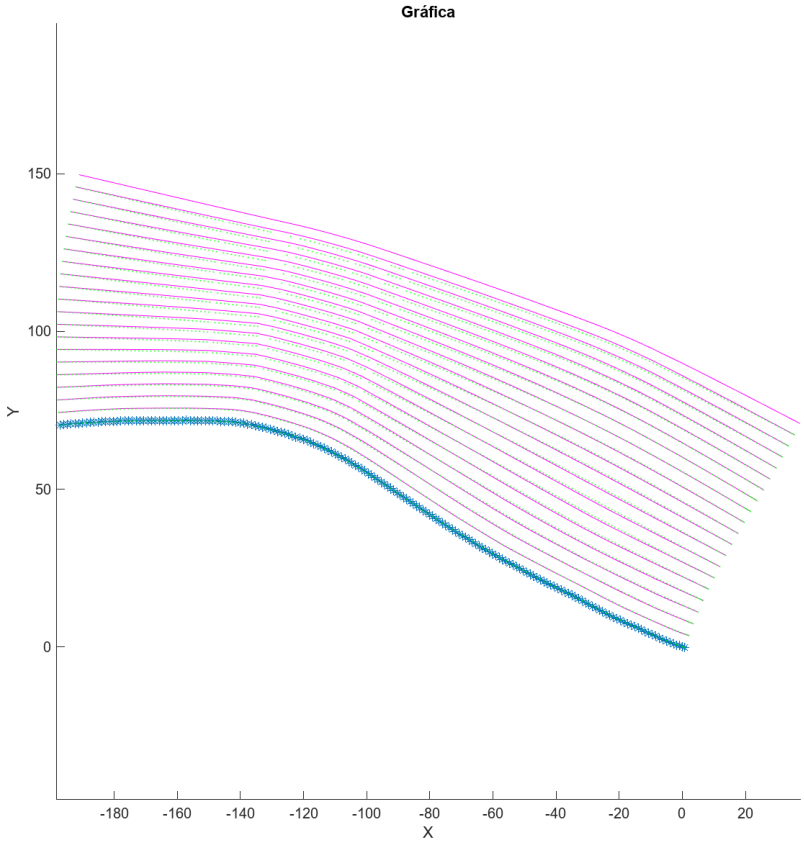

Figura 4.31. Curva 1: Paralelización utilizando los parámetros por defecto que se indican en el inicio del subapartado 4.3. No aparecen cúspides, pero no se elimina completamente la curvatura.

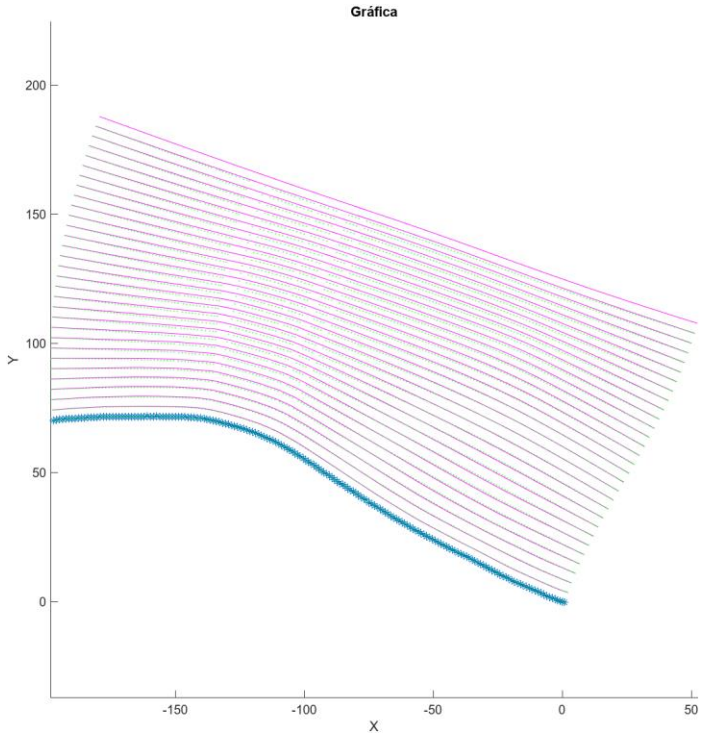

Figura 4.32. Curva 1: Paralelización utilizando los parámetros por defecto que se indican en el inicio del subapartado 4.3 excepto el número de paralelas que se eleva a 30. No aparecen cúspides y se consigue eliminar prácticamente en su totalidad la curvatura.

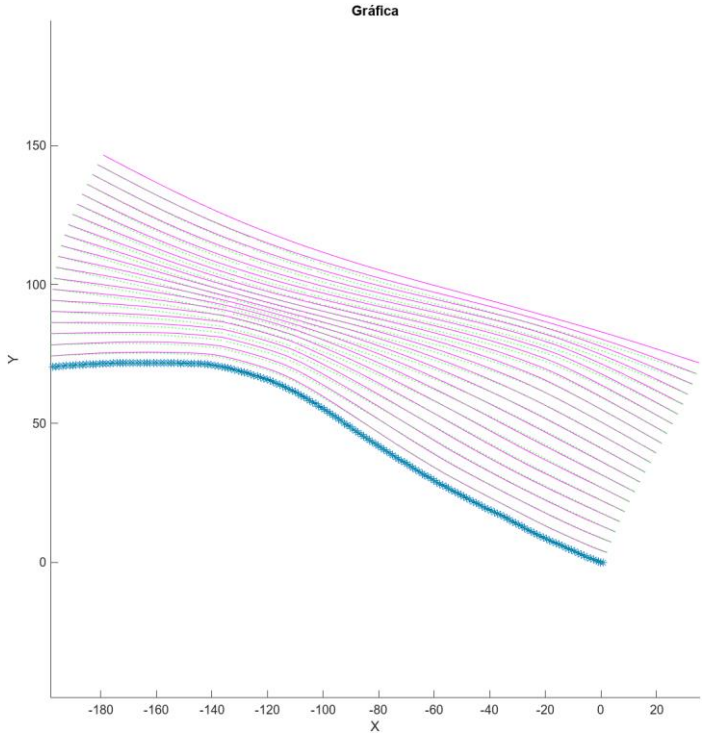

Figura 4.33 Curva 1: Paralelización utilizando los parámetros por defecto que se indican en el inicio del subapartado 4.3 excepto el solape que se eleva a 3. No aparecen cúspides y se consigue eliminar prácticamente en su totalidad la curvatura.

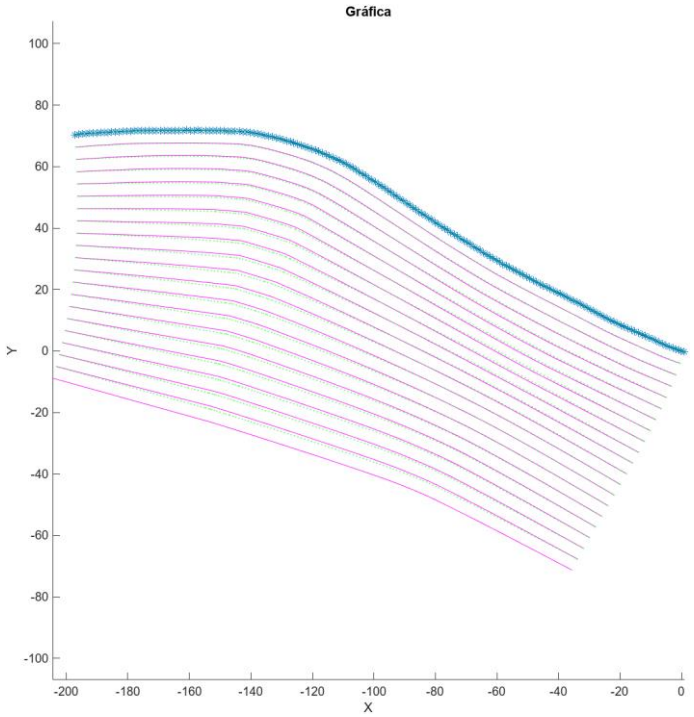

Figura 4.34. Curva 1: Paralelización en dirección negativa. Se utilizan los parámetros por defecto que se indican en el inicio del subapartado 4.3 excepto la distancia de paralela que se cambia de signo. No aparecen cúspides.

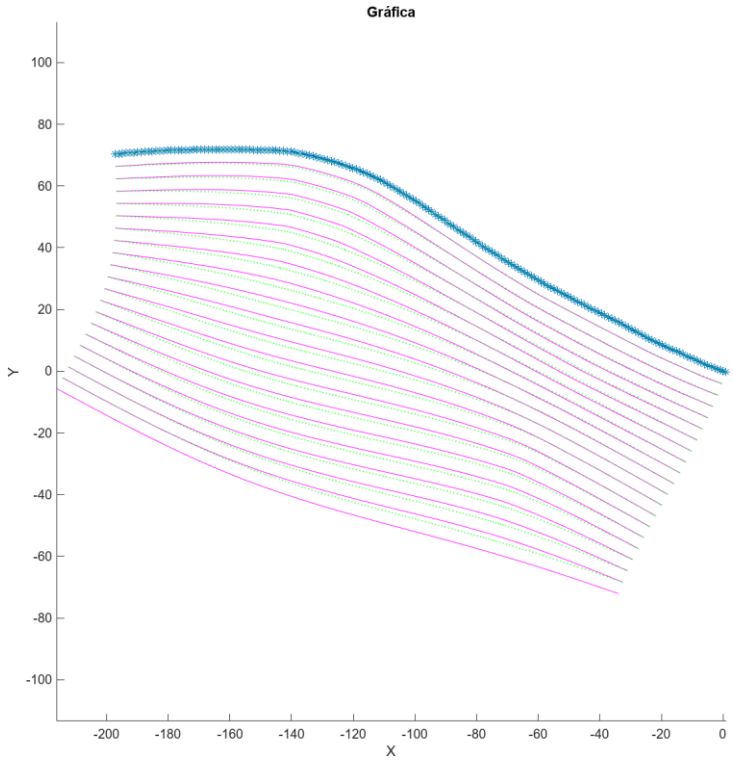

Figura 4.35. Curva 1: Paralelización en dirección negativa y con parámetro sin tratar igual a 3. Se utilizan los parámetros por defecto que se indican en el inicio del subapartado 4.3 excepto la distancia de paralela que se cambia de signo y solape que cambia a 3. No aparecen cúspides y además se consigue un enderezado más rápido que con menor espacio sin tratar.

4.3.2 Curva 2

La segunda curva de la que se presentarán resultados es la que aparece en la **Figura 4.36**. Esta curva tiene un perfil más serpenteado con varios cambios en el sentido de la curvatura. Si generamos las curvas paralelas con los parámetros por defecto obtenemos las curvas que se ven en la **Figura 4.37**. Se puede ver que ha surgido un error, se ha formado una figura similar a una espiral en la parte izquierda de la curva. Esta forma deriva de la aparición de una cúspide que se ha amplificado debido a la ejecución del algoritmo. Como se puede ver en la **Figura 4.38**, la cúspide ocurre en la paralela número ocho. Para evitar la aparición de este fenómeno existen principalmente dos opciones.

Una de ellas consiste en aumentar el espacio que se permite dejar sin tratar para enderezar más cada una de las paralelas y así reducir la curvatura más rápido, es decir, aumentar el parámetro *sin tratar*. Si se configura ese parámetro a dos y se aumenta el umbral a 0.7 no aparecen cúspides y la paralelización se realiza sin problemas, **Figura 4.39.**

En caso de que no sea posible dejar sin tratar el espacio sin tratar de dos metros, la otra opción consiste en aumentar las iteraciones del enderezado. Se aumenta el número de iteraciones del enderezado a ciento cincuenta y el umbral a 0.8. De esta forma se consigue la **Figura 4.40** donde la cúspide ya no aparece y la generación de paralelas se realiza sin problemas.

En la otra dirección la generación de paralelas se realiza sin problemas con los parámetros por defecto siendo el único parámetro modificado la *distancia de paralela* que toma un valor negativo de menos cuatro. Se pueden ver las paralelas en la **Figura 4.41**

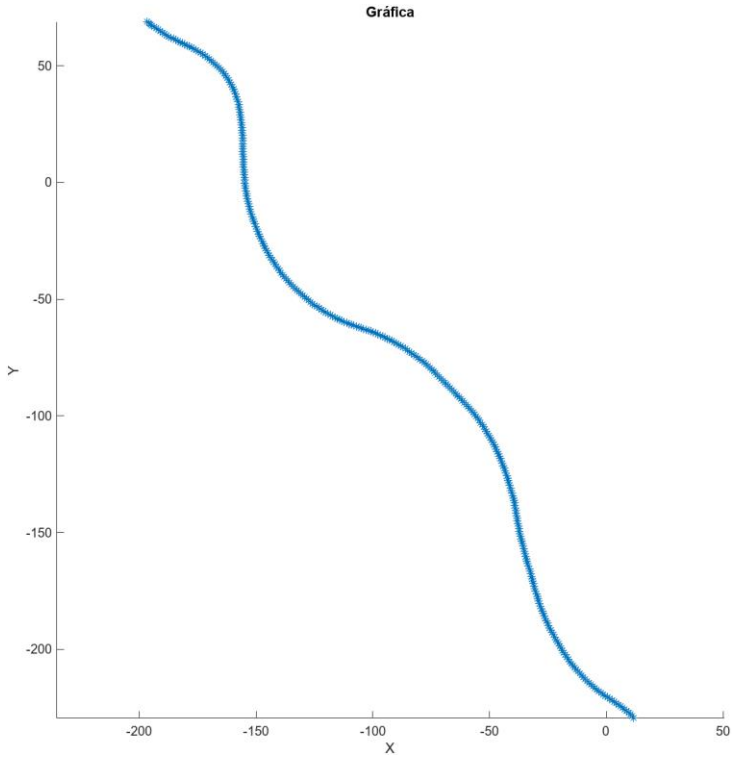

Figura 4.36. Curva 2: Corresponde a un recorrido aleatorio por la parcela.

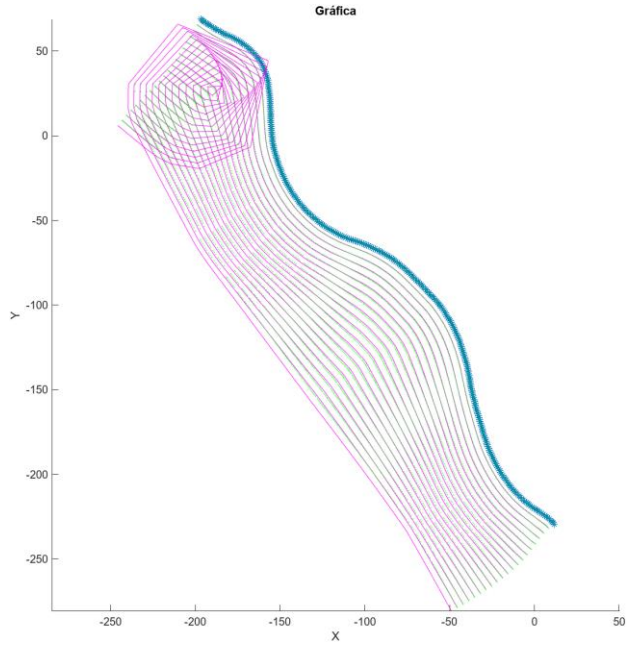

Figura 4.37. Curva 2: Paralelización utilizando los parámetros por defecto que se indican en el inicio del subapartado 4.3. Aparece una espiral en la parte izquierda de la curva que indica la existencia de algún problema en la generación.

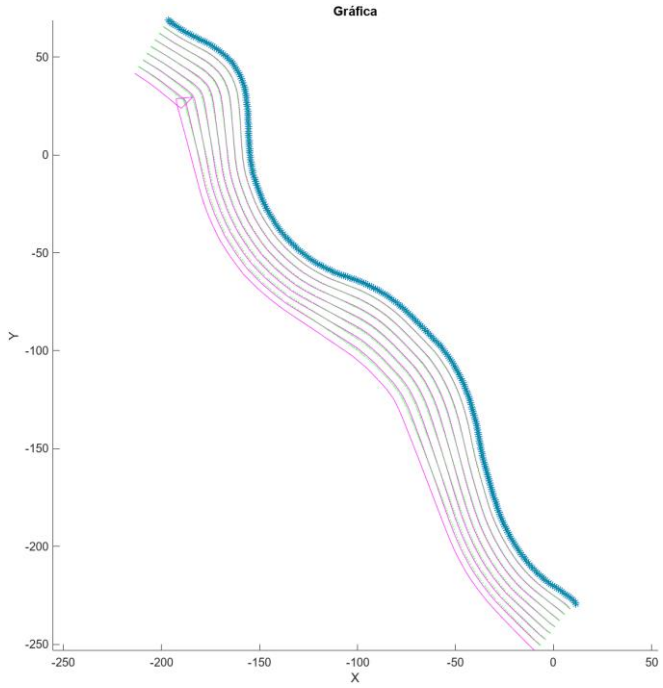

Figura 4.38. Curva 2: Paralelización utilizando los parámetros por defecto que se indican en el inicio del subapartado 4.3 excepto el número de paralelas que se reduce a ocho lo que permite apreciar la cúspide que se produce precisamente en la octava paralela.

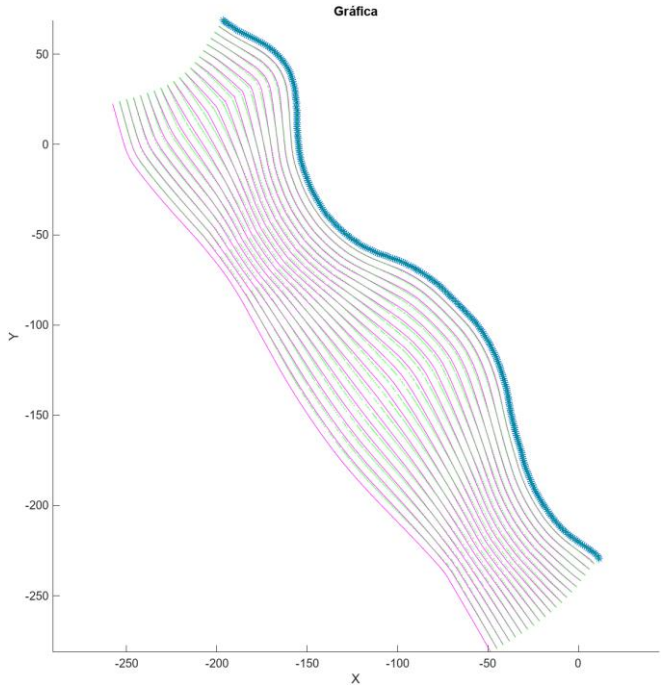

Figura 4.39. Curva 2: Paralelización utilizando los parámetros por defecto que se indican en el inicio del subapartado 4.3 excepto el espacio sin tratar igual a 2 y el umbral igual a 0.7. Estos parámetros hacen que no se produzca la cúspide y la paralelización se realice sin problemas.

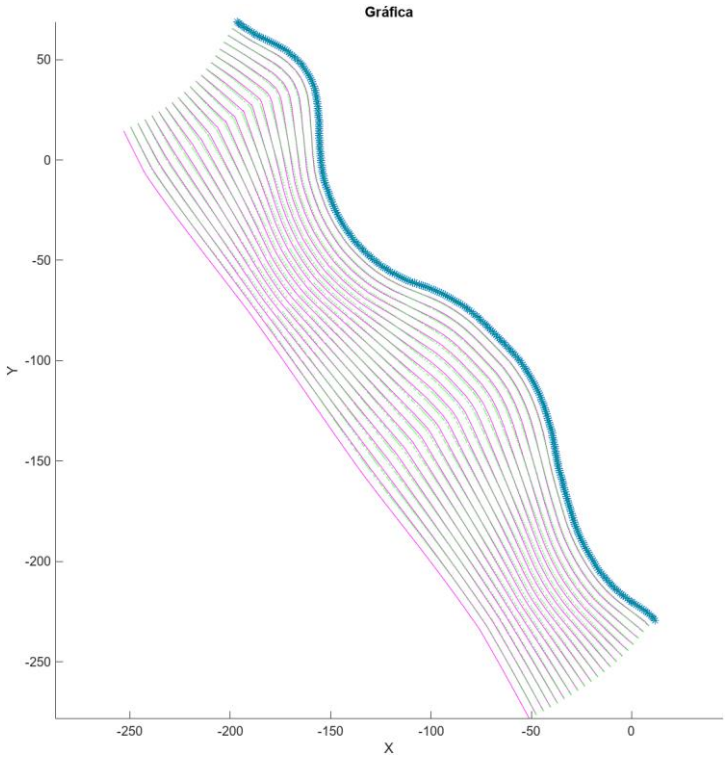

Figura 4.40. Curva 2: *Paralelización utilizando los parámetros por defecto que se indican en el inicio del subapartado 4.3 excepto el número de iteraciones del enderezado igual a 150 y el umbral igual a 0.8. Estos parámetros hacen que no se produzca la cúspide y la paralelización se realice sin problemas.*

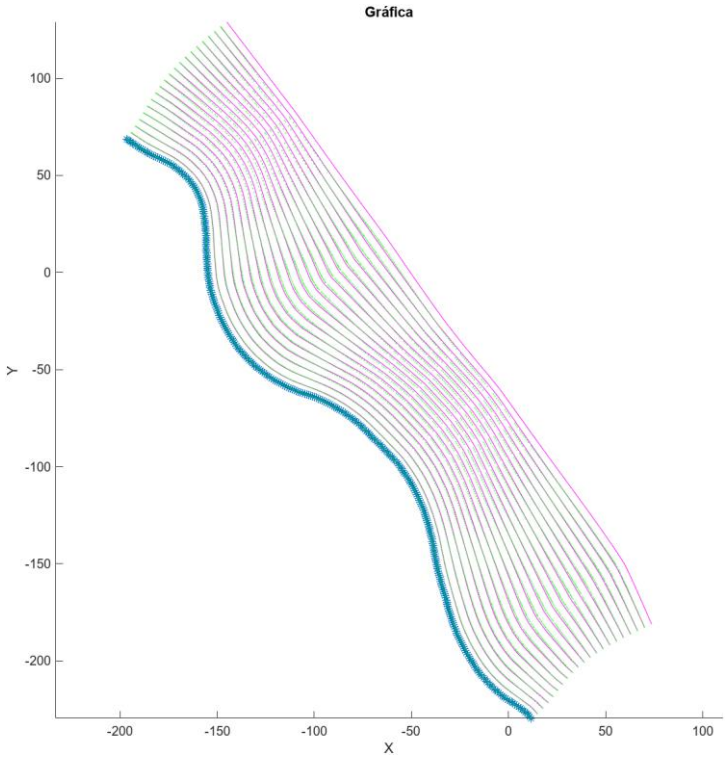

Figura 4.41. Curva 2: Paralelización en dirección negativa. Se utilizan los parámetros por defecto que se indican en el inicio del subapartado 4.3 excepto la distancia de paralela que se cambia de signo. No aparecen cúspides.

4.3.3 Curva 3

La tercera curva de la que se presentarán resultados es la que aparece en la **Figura 4.42**. Si generamos las paralelas con los parámetros por defecto obtenemos las curvas que se ven en la **Figura 4.43**. Se puede ver que ha surgido un error, se ha formado una figura similar a una espiral en la parte inferior de los ejes, la zona donde la curvatura es mayor. Esta forma deriva de la aparición de una cúspide que se ha amplificado debido a la ejecución del algoritmo. Como se puede ver en la **Figura 4.44**, la cúspide ocurre en la paralela número cinco. Para evitar la aparición de este fenómeno se ha encontrado una solución.

La solución consiste en aumentar el espacio que se permite dejar sin tratar para enderezar más cada una de las paralelas y así reducir la curvatura más rápido, es decir, aumentar el parámetro *sin tratar*. Debido a que la curvatura es significativamente grande en la zona, es necesario configurar el parámetro sin tratar con un valor de al menos cuatro. Además, es necesario realizar 150 iteraciones, con un umbral de tres y el doble de puntos dinámicos que por defecto para tener más margen de elección y acelerar el enderezado por curva. El resultado de aplicar estos parámetros se ve en la **Figura 4.45**.

Los parámetros anteriores no consiguen eliminar por completo la curvatura. Cambiando el solape a dos se puede conseguir una mejor aproximación a la recta, **Figura 4.46**.

Debido a que es necesario dejar hasta 4 metros de espacio sin tratar para evitar la generación de la cúspide, es posible que en este caso no se pueda llevar a la práctica. En el estado actual de la aplicación no se ha encontrado otra solución que permita eliminar la cúspide con menos de 4 en el parámetro *sin tratar*.

En la otra dirección de generación de paralelas, utilizando los parámetros por defecto excepto con *distancia paralela* negativa, también aparece una cúspide. Aparece en la paralela número nueve como se puede ver en la **Figura 4.47**. Para evitar la aparición de este fenómeno existen principalmente dos soluciones similares a las utilizadas en la curva 2.

La primera consiste en aumentar el espacio que se permite dejar sin tratar para enderezar más cada una de las paralelas y así reducir la curvatura más rápido, es decir, aumentar el parámetro *sin tratar*. Si se configura ese parámetro a dos y se aumenta el umbral a 0.7 no aparecen cúspides y la paralelización se realiza sin problemas, **Figura 4.48**. Se han generado 30 paralelas para que se aprecie mejor que no aparece ningún error en la generación de estas curvas.

En caso de que no sea posible dejar sin tratar el espacio sin tratar de dos metros, la otra opción consiste en aumentar las iteraciones del enderezado. Se aumenta el número de iteraciones del enderezado a 150 y el umbral a 1. De esta forma se consigue la **Figura 4.49** donde la cúspide ya no aparece y la generación de paralelas se realiza sin problemas. Hay que destacar que se han generado 35 curvas para ver mejor la generación de paralelas sin errores. Aumentando, *solape* o *sin tratar* se consigue un enderezado más rápido y, por lo tanto, en menos curvas.

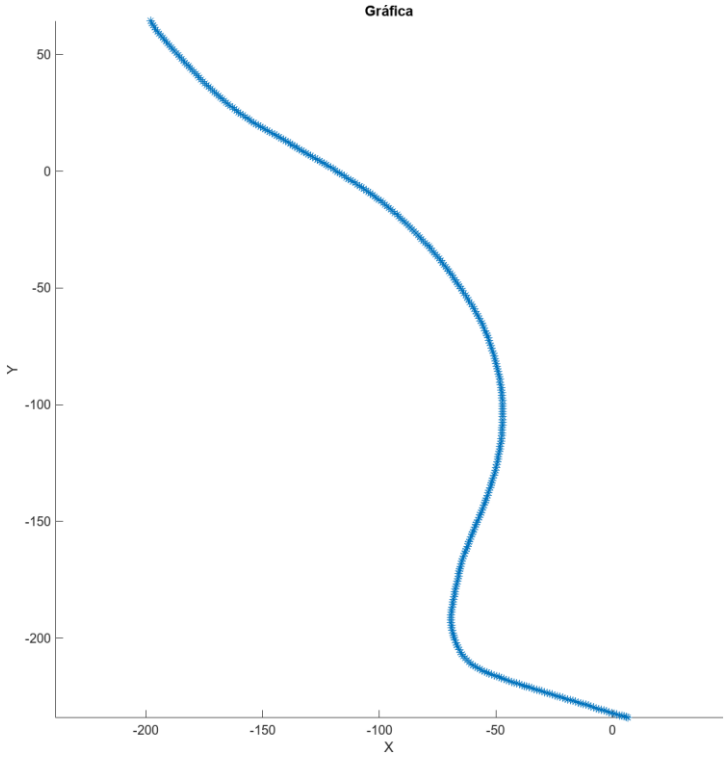

Figura 4.42. Curva 3: Corresponde a un recorrido aleatorio por la parcela.

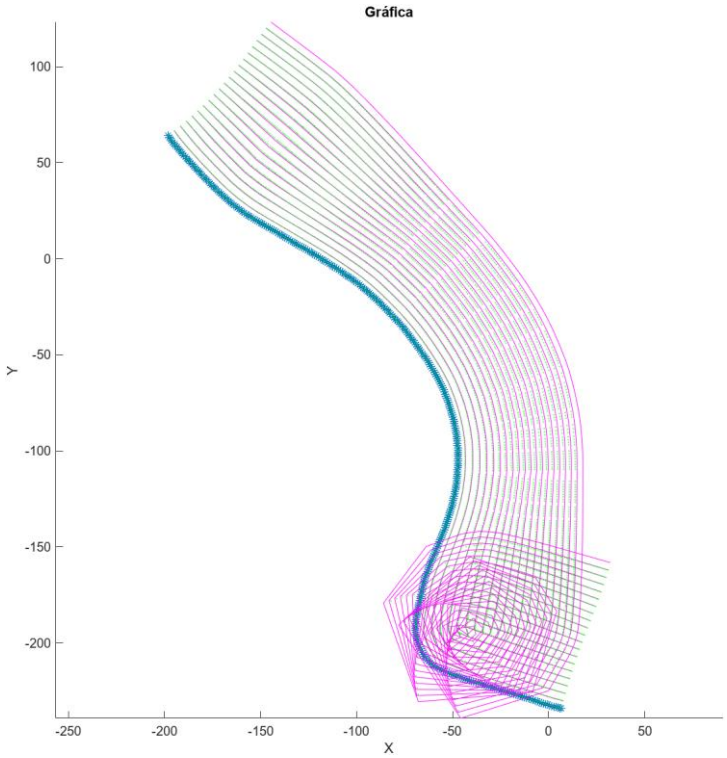

Figura 4.43. Curva 3: Paralelización utilizando los parámetros por defecto que se indican en el inicio del subapartado 4.3. Aparece una espiral en la parte inferior derecha de la curva que indica la existencia de algún problema en la generación de las paralelas

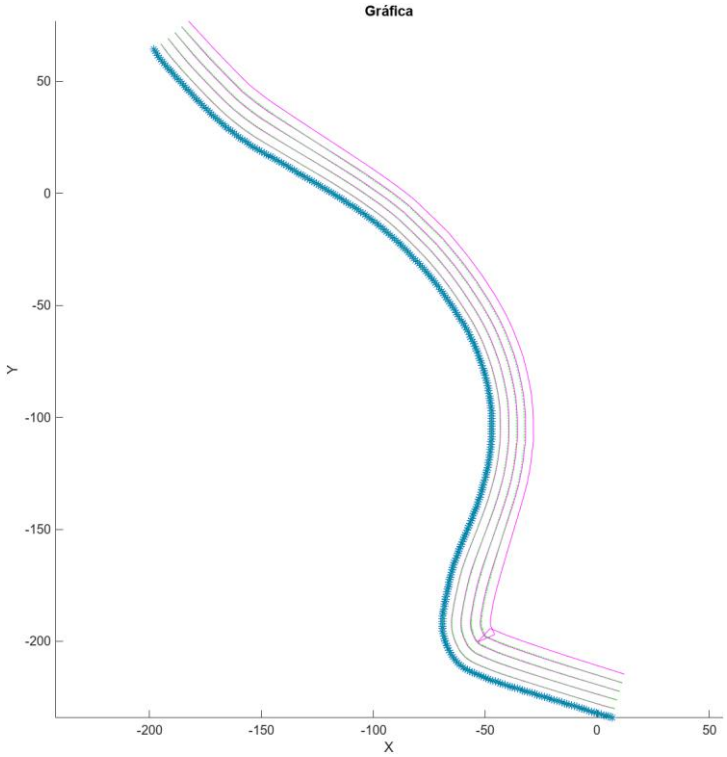

Figura 4.44. Curva 3: Paralelización utilizando los parámetros por defecto que se indican en el inicio del subapartado 4.3 excepto el número de paralelas que se reduce a nueve lo que permite apreciar la cúspide que se produce precisamente en la novena paralela.

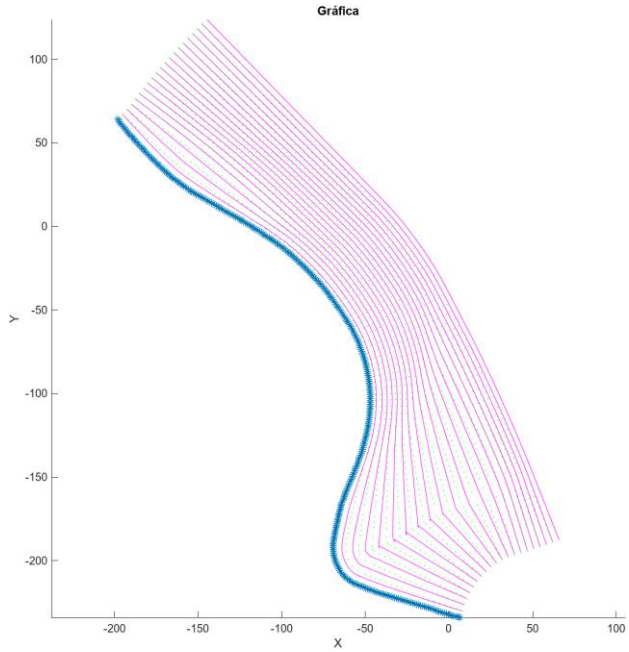

Figura 4.45. Curva 3: Paralelización utilizando los parámetros por defecto que se indican en el inicio del subapartado 4.3 excepto el número de iteraciones del enderezado igual a 150, el umbral igual a 3, el espacio sin tratar igual a 4 y con 40 puntos dinámicos. Estos parámetros hacen que no se produzca la cúspide, aunque el uso de estos valores haga que no sea práctico para utilizar en el mundo real en este caso

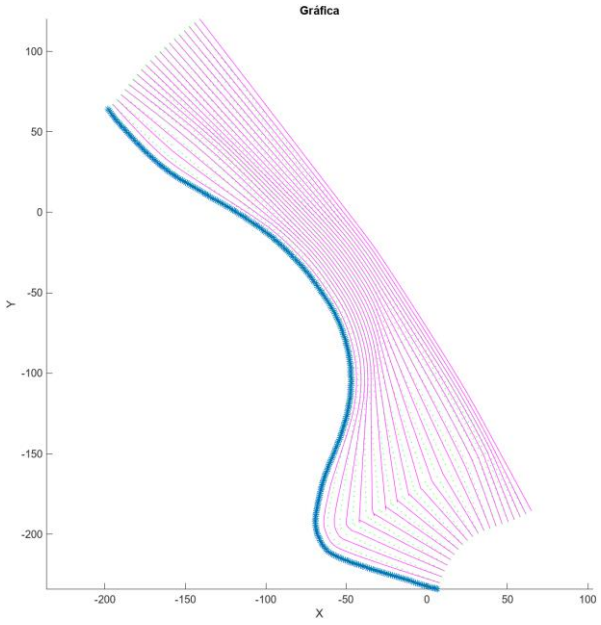

Figura 4.46. Curva 3: Paralelización utilizando los parámetros por defecto que se indican en el inicio del subapartado 4.3 excepto el número de iteraciones del enderezado igual a 150, el umbral igual a 3, el espacio sin tratar igual a 4, solape igual a 2 y con 40 puntos dinámicos. Estos parámetros hacen que no se produzca la cúspide y, al permitir más solape, que se reduzca más la curvatura. El uso de estos valores haga que no sea práctico para utilizar en el mundo real en este caso

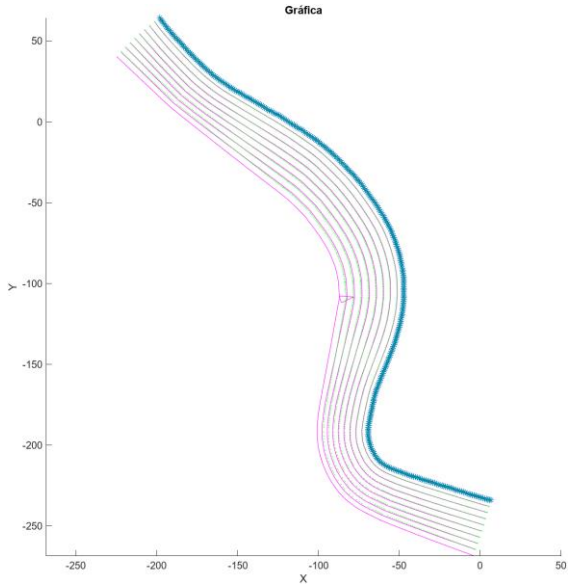

Figura 4.47. Curva 3: Paralelización en dirección negativa. Se utilizan los parámetros por defecto que se indican en el inicio del subapartado 4.3 excepto el número de paralelas que se reduce a nueve y la distancia de paralela que se cambia de signo. Permite apreciar la cúspide que se produce precisamente en la novena paralela.

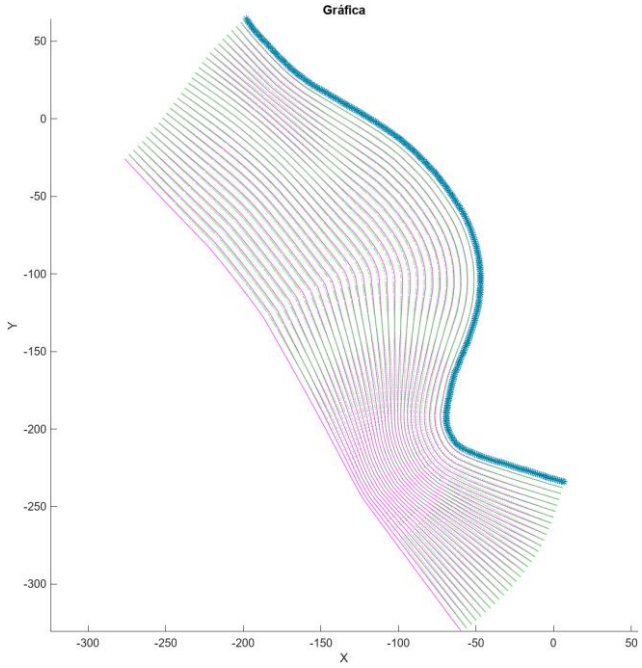

Figura 4.48. Curva 3: Paralelización en dirección negativa. Se utilizan los parámetros por defecto que se indican en el inicio del subapartado 4.3 excepto la distancia de paralela que se cambia de signo, el espacio sin tratar igual a 2 y el umbral igual a 0.7. Estos parámetros hacen que no se produzca la cúspide y la paralelización se realice sin problemas.

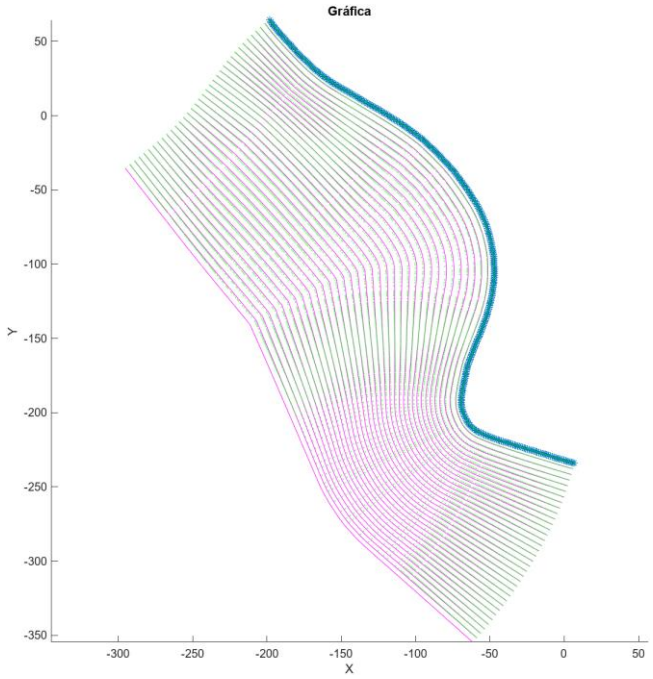

Figura 4.49. Curva 3: Paralelización en dirección negativa. Se utilizan los parámetros por defecto que se indican en el inicio del subapartado 4.3 excepto la distancia de paralela que se cambia de signo, el número de iteraciones del enderezado igual a 150 y el umbral igual a 1. Estos parámetros hacen que no se produzca la cúspide y la paralelización se realice sin problemas.

4.3.4 Curva 4

La cuarta curva de la que se presentarán resultados es la que aparece en la **Figura 4.50**. Si generamos las curvas paralelas con los parámetros por defecto obtenemos las curvas que se ven en la **Figura 4.51**. Se puede ver que ha surgido un error, se ha formado una figura similar a una espiral en la parte superior derecha de los ejes. Esta forma deriva de la aparición de una cúspide que se ha amplificado debido a la ejecución del algoritmo. Como se puede ver en la **Figura 4.52**, la cúspide ocurre en la decimotercera paralela. Para evitar la aparición de este fenómeno existen principalmente dos opciones.

La primera consiste en aumentar el espacio que se permite dejar sin tratar para enderezar más cada una de las paralelas y así reducir la curvatura más rápido, es decir, aumentar el parámetro *sin tratar*. Si se configura ese parámetro a 2 y se aumenta el umbral a 0.7 no aparecen cúspides y la paralelización se realiza sin problemas, **Figura 4.53**. Al igual que para la curva 3, se han generado 30 paralelas para que se aprecie mejor que no aparece ningún error en la generación de estas curvas.

En caso de que no sea posible dejar sin tratar el espacio sin tratar de dos metros, la otra opción consiste en aumentar las iteraciones del enderezado. Se aumenta el número de iteraciones del enderezado a 150 y el umbral a 1.2. De esta forma se consigue la **Figura 4.54** donde la cúspide ya no aparece y la generación de paralelas se realiza sin problemas.

En la otra dirección la generación de paralelas se realiza sin problemas con los parámetros por defecto excepto el umbral que se configura con 0.6 y el número de curvas que se eleva a 30 para ilustrar más completo el proceso de enderezado. Además, el parámetro *distancia de paralela* toma un valor negativo de menos cuatro. Se pueden ver las paralelas en la **Figura 4.55**.

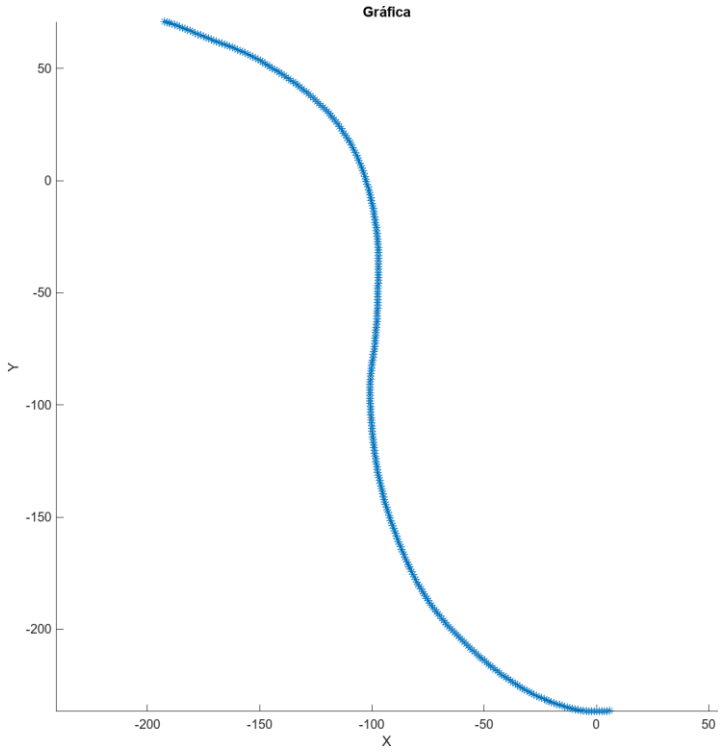

Figura 4.50. Curva 4: Corresponde a un recorrido aleatorio por la parcela.

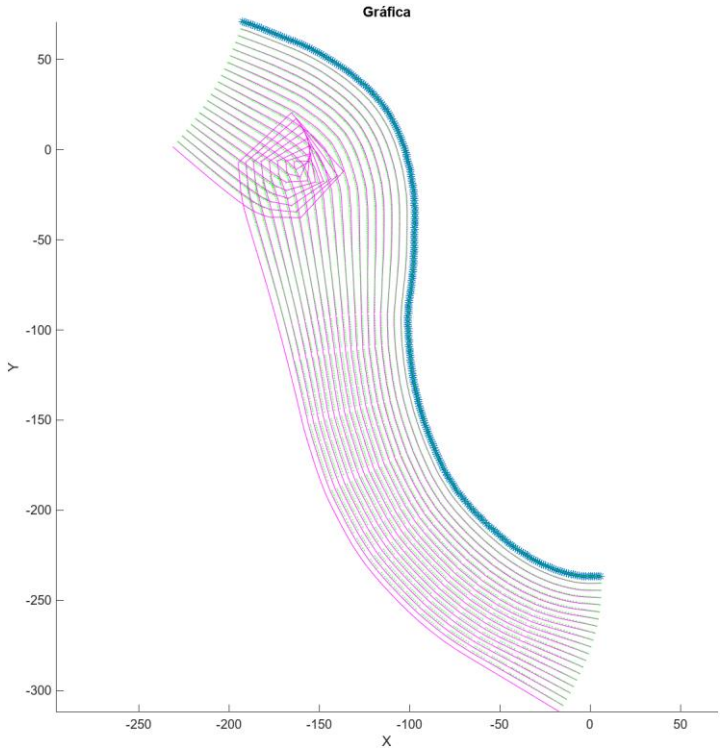

Figura 4.51. Curva 4: Paralelización utilizando los parámetros por defecto que se indican en el inicio del subapartado 4.3. Aparece una espiral en la parte superior izquierda de la curva que indica la existencia de algún problema en la generación de las paralelas.

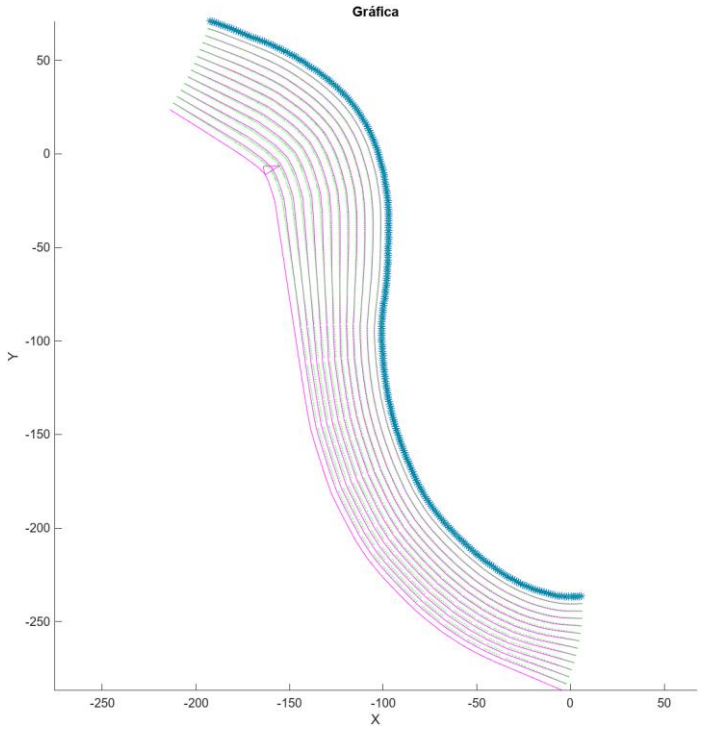

Figura 4.52. Curva 4: Paralelización utilizando los parámetros por defecto que se indican en el inicio del subapartado 4.3 excepto el número de paralelas que se reduce a trece lo que permite apreciar la cúspide que se produce precisamente en la decimotercera paralela.

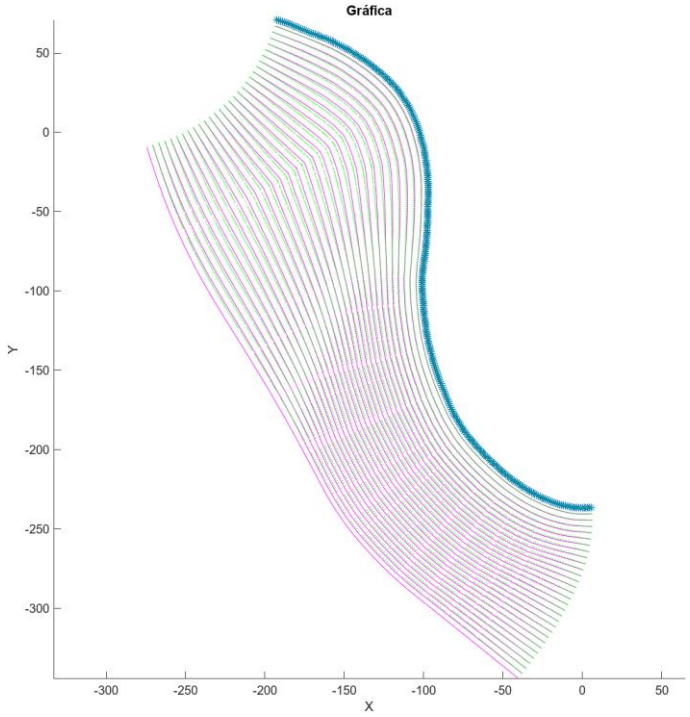

Figura 4.53. Curva 4: Paralelización utilizando los parámetros por defecto que se indican en el inicio del subapartado 4.3 excepto el espacio sin tratar igual a 2 y el umbral igual a 0.7. Estos parámetros hacen que no se produzca la cúspide y la paralelización se realice sin problemas.

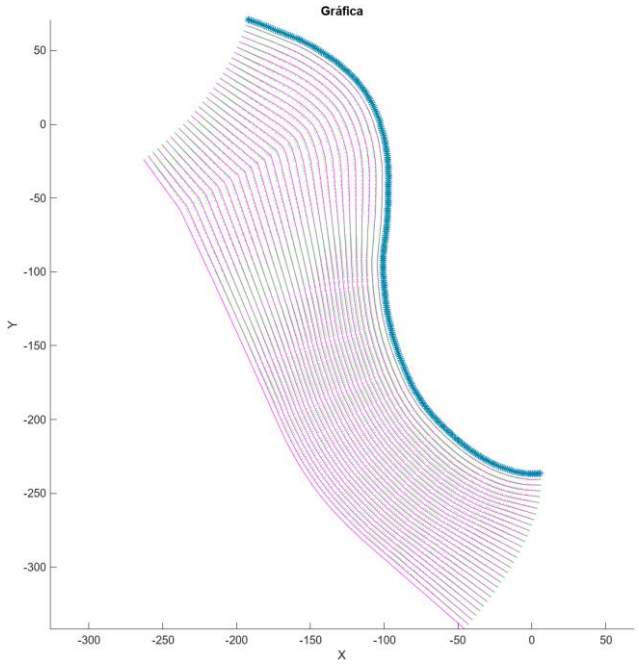

Figura 4.54. Curva 4: Paralelización utilizando los parámetros por defecto que se indican en el inicio del subapartado 4.3 excepto el número de iteraciones del enderezado igual a 150 y el umbral igual a 1.2. Estos parámetros hacen que no se produzca la cúspide y la paralelización se realice sin problemas.

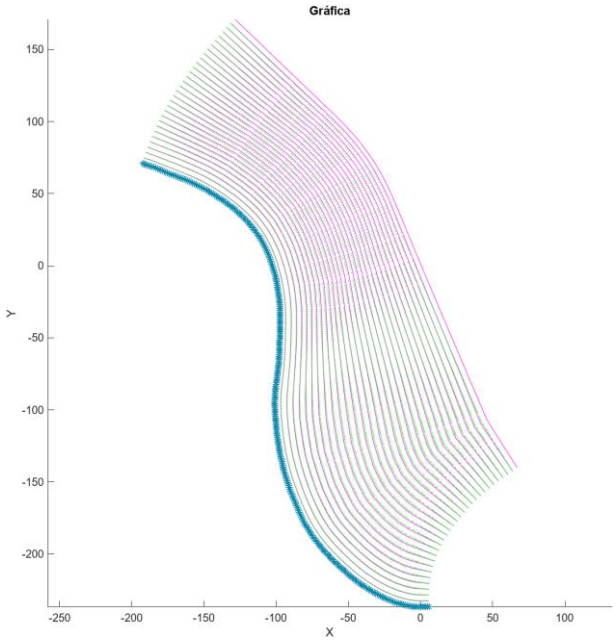

Figura 4.55. *Paralelización en dirección negativa. Se utilizan los parámetros por defecto que se indican en el inicio del subapartado 4.3 excepto la distancia de paralela que se cambia de signo y el umbral que pasa a valer 0.7. Estos parámetros hacen que no se produzca una cúspide y la paralelización se realice sin problemas.*

4.4 Conclusiones

En este capítulo se ha descrito la interfaz y el código de la aplicación diseñada. También se han presentado diferentes curvas paralelizadas. Con los parámetros por defecto tres de las cuatro curvas presentaban cúspides en una dirección de paralelización o, en el caso de la curva 3 y 4, en ambas

direcciones. Para corregir estos problemas se han presentado diferentes soluciones consistentes en la modificación de los valores de parámetros. Se ha visto que la aplicación funciona adecuadamente para corregir curvas en las que no hay demasiada curvatura. Para curvas con una gran curvatura se requieren soluciones más bruscas en la elección de los valores de los parámetros, pero finalmente consigue realizar la paralelización y el enderezado de forma adecuada.

Capítulo 5: Conclusiones y líneas futuras

Capítulo 5: Conclusiones y líneas futuras

En este trabajo se han estudiado diferentes métodos de generación de curvas paralelas. Los métodos matemáticos son más complejos que los métodos algorítmicos usando librerías, pero tienen la ventaja de que dan más versatilidad a la hora de implementarlos en un lenguaje de programación. La implementación, que utiliza funciones de librerías, tiene una principal desventaja. El manejo de sus funciones depende de la implementación que haya realizado el creador de la librería lo que hace que este método sea más limitado e inflexible en algunos aspectos. Otro punto para destacar es que, normalmente la paralela generada tiene una expresión mucho más compleja que la curva normal. Para la creación del algoritmo se ha utilizado el método de la paralela a partir de vector normal.

En la generación de paralelas por el método de paralelización a partir de vector normal aparecen discontinuidades. Estas discontinuidades reciben el nombre de cúspides. Además de causar la pérdida de forma de las paralelas, estas discontinuidades provocan que, en un contexto real, la curva sea imposible de seguir por el tractor en el guiado debido a los cambios bruscos de dirección que se necesitarían hacer. Por ello es necesaria una mejora del algoritmo.

Se ha propuesto y descrito un algoritmo de generación de paralelas a partir de vector normal combinado con enderezado de curvas. Este algoritmo es bastante iterativo e intenta buscar la mejor curva a fuerza bruta probando diferentes posiciones para los puntos. Con este algoritmo se diseñó una aplicación MATLAB. Los parámetros de la aplicación son muy sensibles y cualquier cambio en alguno de ellos produce variaciones en la generación de las paralelas. Además, el código utiliza un bucle principal de paralelización que a su vez tiene el bucle de enderezado anidado. Esto hace que el proceso de ejecución sea muy iterativo pudiendo provocar problemas de rendimiento. En concreto, aumentar en exceso los parámetros *iteraciones del enderezado* y *Puntos Dinámicos* provoca que el rendimiento empeore significativamente.

En las paralelas generadas se vio como estas se podían generar tanto con un espacio sin tratar mayor que el de solape, para las curvas 2, 3 y 4 en caso de que no haya tanto espacio para solapar disponible por el tipo aplicación agraria a realizar. También ocurre lo contrario, se puede dejar un espacio de solape mayor que el espacio sin tratar como se vio en la curva 1. También hay casos en los que ambos valores pueden ser similares.

Como se vio en la presentación de los resultados en ocasiones puede ser necesario hacer grandes variaciones en los valores de los parámetros cuando la curvatura es muy grande en alguna zona haciendo incluso que no sea posible llevarlo al terreno práctico real. A pesar de esto, la aplicación es capaz de realizar la paralelización sin problemas para las cuatro curvas ajustando ligeramente los parámetros. En función de los resultados se pueden recomendar algunos valores para los parámetros de forma general. Para curvas con una curvatura no demasiado alta suele ser suficiente con establecer

150 iteraciones o incluso menos, aunque no es recomendable menos de 100. Los puntos dinámicos se pueden configurar entre 20 y 40. No es recomendable añadir más por cuestiones de rendimiento. Los números de puntos de solape y sin tratar, así como los parámetros de *Exactitud Solape* y *Exactitud sin tratar* no se recomiendan modificar porque no afectan demasiado al resultado de la paralelización y pueden llegar a afectar fuertemente al rendimiento. El umbral es el parámetro más variable entre curvas, puede valer desde 0.4 a 1.2 en los casos normales y hasta 3 cuando la curvatura es demasiado grande.

En base a lo comentado anteriormente se abren bastantes líneas de investigación futuras y de mejora. El siguiente objetivo inmediato pasa por mejorar el proceso de enderezado intentando mejorar su velocidad y rendimiento. Para ello, por ejemplo, se podría recurrir a variar el algoritmo intentando eliminar o reducir el bucle o cambiar la función *M* buscando otra que tenga un comportamiento más adecuado. Otra opción sería buscar otro algoritmo diferente de enderezado más eficaz, aunque pudiese ser más complejo. También se podrían explorar otras técnicas de generación de paralelas que tiendan a generar menos cúspides. Otra línea de investigación futura podría intentar usar splines, cuyas aplicaciones no se han tratado en este trabajo.

Referencias

[1] Sean Gillies. Shapely 2.0.3 documentation. https://shapely.readthedocs.io/en/stable/manual.html#introduction. Último acceso: 27/02/2024

[2] Wikipedia contributors. Parallel curve. https://en.wikipedia.org/wiki/Parallel_curve. Último acceso: 14/09/2024

[3] By Ag2gaeh – Own work, CC BY-SA 4.0, https://commons.wikimedia.org/wiki/File:Offset-definition-poss.svg. Último acceso: 04/09/2024

[4] By Ag2gaeh – Own work, CC BY-SA 4.0, https://commons.wikimedia.org/wiki/File:Offset-definition-poss.svg. Último acceso: 14/09/2024

[5] Sean Gillies. Shapely 2.0.3 documentation https://shapely.readthedocs.io/en/stable/reference/shapely.buffer.html#shapely.buffer. Último acceso: 05/03/2024

[6] Lee, I., Kim, M., & Elber, G. (1998). Polynomial/Rational Approximation of Minkowski Sum Boundary Curves. Graphical Models And Image Processing, 60(2), 136-165.

[7] Martin Davis. GEOS 3.14.0dev documentation https://libgeos.org/doxygen/classgeos_1_1operation_1_1buffer_1_1OffsetCurve.html. Último acceso: 07/05/2024

[8] Farouki, R.T., Neff, C.A., 1990. Analytic properties of plane offset curves. Computer Aided Geometric Design 7, 83–99.

[9] © 1994-2024 The MathWorks, Inc. MATLAB App Designer https://es.mathworks.com/products/matlab/app-designer.html. Último acceso: 02/09/2024

[10] Desmos Calculator. https://www.desmos.com/calculator/1i6kzmbear. Último acceso: 10/09/2024

[11] Desmos Calculator. https://www.desmos.com/calculator/o0q5ffgpyh. Último acceso: 10/09/2024

[12] © 1994-2024 The MathWorks, Inc. Suma de los elementos de un arreglo (sum) https://es.mathworks.com/help/matlab/ref/sum.html. Último acceso